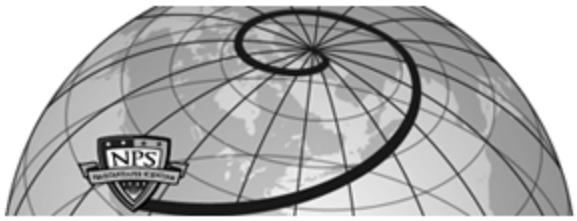
2000-08

Design of a video recording and tracking system for the NPS Autonomous Underwater Vehicle (AUV)

Dussourd, Xavier

Monterey, California. Naval Postgraduate School



NPS-ME-00-003

NAVAL POSTGRADUATE SCHOOL

Monterey, California

Design of a Video Recording and Tracking System for the NPS Autonomous Underwater Vehicle (AUV)

by

Xavier Dussourd

Ecole Nationale d'Ingenieurs de Tarbes (ENIT), France

August 2000

TECHNICAL REPORT

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93940-5000

RADM Richard H. Wells, USNR
Superintendent

Richard Elster
Provost

This report was prepared for the Center for Autonomous Underwater Vehicle (CAUVR) and funded by the Office Naval Research (ONR) (Dr. Tom Curtin) under project N° N0001498wr30175

This report was prepared by:

[redacted] 7 August 2000

Xavier Dussourd
Visiting Student, Ecole Nationale d'Ingenieurs de Tarbes (ENIT), France

Reviewed by:

[redacted] 7 AUGUST 2000

Assistant Professor Don Brutzman

Released by:

[redacted]

Professor Tony Healey

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** August 2000	**3. REPORT TYPE AND DATES COVERED** Technical Report, February-August 2000	
4. TITLE AND SUBTITLE: DESIGN OF A VIDEO RECORDING AND TRACKING SYSTEM FOR THE NPS AUTONOMOUS UNDERWATER VEHICLE (AUV)			**5. FUNDING NUMBERS**
6. AUTHOR Xavier Dussourd			
7. PERFORMING ORGANIZATION NAME AND ADDRESS Code UW/Br, Undersea Warfare Academic Group Naval Postgraduate School Monterey, CA 93943-5000			**. PERFORMING ORGANIZATION EPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research 800 North Quincy St. Arlington, VA 22217			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Distribution Statement (mix case letters)			**12b. DISTRIBUTION CODE**

13. ABSTRACT

The Naval Postgraduate School (NPS) is on the leading edge of Autonomous Underwater Vehicle (AUV) research and is currently developing a new AUV named ARIES. AUVs are principally designed to realize reconnaissance missions. To achieve such tasks, they usually utilize precise range-based sonars. Unfortunately, sonars are often unable to provide a good and efficient description of their environment due to the unclear aspect of many sonar images. To overcome such problems, the ARIES vehicle has a video camera to obtain understandable, classifiable images. This project explains the design of the video system installed on the ARIES AUV. The video system allows the AUV to automatically record and to easily understand the images provide by the AUV's camera. In this way, the AUV is now an efficient "witness" of its surroundings. This project then describes how to utilize the video for vehicle navigation. Such a capability is demonstrated by the simulation of the recovery between an AUV and a host submarine, using an underwater beacon and a light-tracking system. These results show that the use of in-water video for contact recording and navigational tracking can be practical and effective.

14. SUBJECT TERMS Autonomous Underwater Vehicle, AUV, video system, image synthesis, sonar, tracking, virtual worlds			**15. NUMBER OF PAGES** 95
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)

Prescribed by ANSI Std. 239-18

ABSTRACT

The Naval Postgraduate School (NPS) is on the leading edge of Autonomous Underwater Vehicle (AUV) research and is currently developing a new AUV named ARIES.

AUVs are principally designed to realize reconnaissance missions. To achieve such tasks, they usually utilize precise range-based sonars. Unfortunately, sonars are often unable to provide a good and efficient description of their environment due to the unclear aspect of many sonar images. To overcome such problems, the ARIES vehicle has a video camera to obtain understandable, classifiable images.

This project explains the design of the video system installed on the ARIES AUV. The video system allows the AUV to automatically record and to easily understand the images provide by the AUV's camera. In this way, the AUV is now an efficient "witness" of its surroundings. This project then describes how to utilize the video for vehicle navigation. Such a capability is demonstrated by the simulation of the recovery between an AUV and a host submarine, using an underwater beacon and a light-tracking system.

These results show that the use of in-water video for contact recording and navigational tracking can be practical and effective.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to all the members of the NPS AUV Research Group.

Firstly, I like to thank Professor Tony Healey and Assistant Professor Don Brutzman who welcomed me and gave me unconditional enthusiastic support throughout this project.

I'm also very grateful to Dr. David Marco who provided as much help as I needed. His guidance and knowledge have been indispensable for the outcome of my work.

A special thanks to Olivier Doucy for his great advice and time he provided during the project.

I would also like to thank Robert Dayap for his aid during my stay.

Lastly, but mostly, I would like to thank my project supervisor in France, Thierry Vidal, who has given me the opportunity to live this wonderful experience abroad.

I. INTRODUCTION

A. OVERVIEW

An Autonomous Underwater Vehicle (AUV) operates independently of any physical or electrical tether. It carries all its computerized commands with it and can have sophisticated sensors that collect information. This type of vehicle is designed to reach places where people can't or (shouldn't) go.

One of the main objectives in the design of an AUV is to perform observation and reconnaissance missions. To achieve this goal, the AUV is equipped with sonar. However the range-based images provided by sonar are often not legible enough to be clearly understood for classifying discovered objects. For example, detecting the difference between a mine and a refrigerator is not so easy using sonar images. To overcome this problem, the NPS AUV research team decided to add a video camera to the ARIES vehicle so that the AUV might become a good "witness" of its environment.

In order to be a powerful and useful tool, the video camera needs to be integrated as part of a complete system providing a really capable instrument for an AUV. This system should allow recording, obtaining and interpreting the video images. Once equipped with a video camera, it is interesting to use this new sensor not only as a passive device but also as a navigational system helping the vehicle in its motion control. This second utilization is studied through a light-tracking system simulation.

The motivation for this study is to design an efficient video system, and to demonstrate through simulation that the video can also be used as a sensor for the vehicle guidance.

B. ORGANIZATION OF THE REPORT

This report is organized into four chapters. Chapter II provides a general overview of the ARIES NPS AUV, which is the vehicle actually developed. System concepts, hardware, and software are detailed there.

Chapter III presents the designed video system. Each component is described and their implementations explained.

Chapter IV presents the simulation of the guidance system using the video. The system simulated is described, the simulation software explained, and results and validation are provided.

Research conclusions and recommendations for future works are provided in the Chapter V. Appendices provide details regarding the device specifications and software modifications.

II. RELATED WORK

A. INTRODUCTION

The Center for Autonomous Underwater Vehicle (AUV) Research of the Naval Postgraduate School of Monterey has been active in developing Autonomous Underwater Vehicle Technology since 1987. Missions for these vehicles include ocean data gathering, monitoring, and other such as mine hunting and neutralization. The actual vehicle, the NPS ARIES AUV, is the third generation AUV from the NPS Center for AUV Research. This chapter provides general overview of the ARIES vehicle, focusing on hardware and software.

B. AUV ARIES PRESENTATION

The NPS ARIES AUV construction started in 1998. It is designed to be an operational vehicle in shallow water to 100 meters depth and its main goals are mine reconnaissance and multi-vehicle communications.

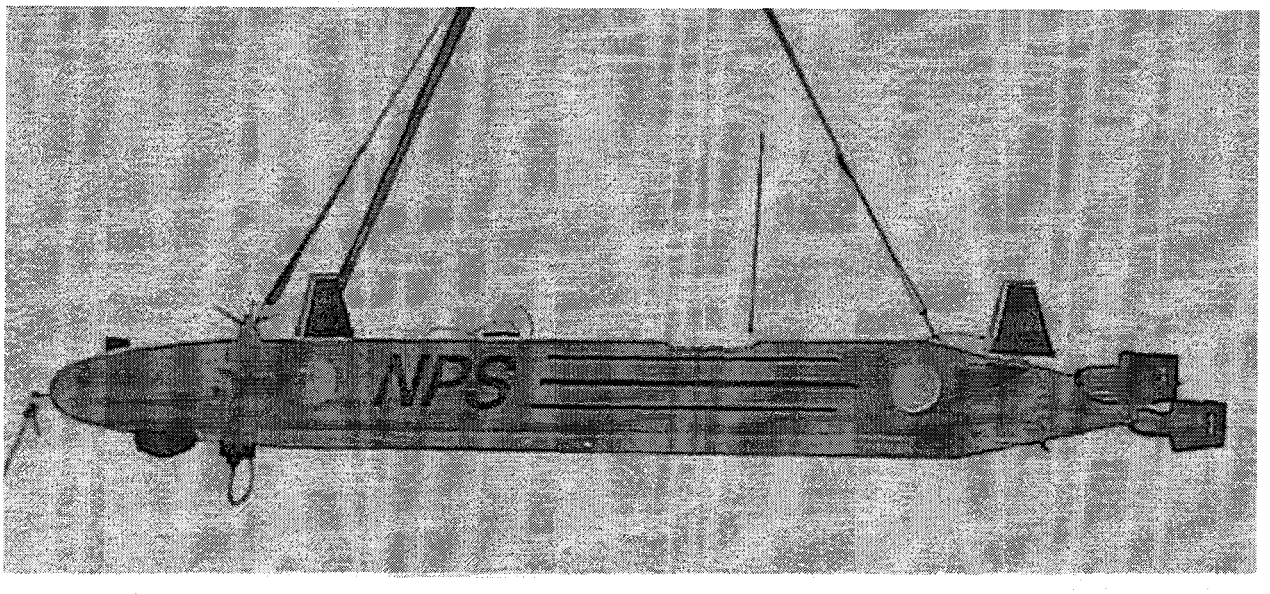


Figure II.1: The NPS ARIES AUV on the hook at AUVFest '99

1. Hardware

The ARIES is a complex robot containing various motors, controllers, servo-amplifiers and computer in a watertight hull. An internal view of the hardware layout is shown below in Figure II.2.

The vehicle weights 225kg and the aluminum hull measure approximately 3m long, 0.4m wide and 0.25 m high. It has a free-flooding nose cone that houses some of the AUV's measurement devices. It is capable of a top speed of 3.5 knots and is powered by six 12 V rechargeable lead acid batteries. The endurance is approximately four hours at top speed, twenty hours hotel load only.

Main propulsion is realized using twin electric thrusters located at the stern. Heading and depth is controlled using upper bow and stern rudders and a set of bow and stern planes. For very slow (or zero) forward-speed maneuvers, vertical and lateral cross-body thrusters are used to control the vehicle posture.

Four sensors are used for the navigation:

- A RDI Doppler that measures the vehicle ground speed and altitude.
- A TMC2 magnetic compass that measures the vehicle magnetic heading.
- A Systron Donner 3-axis Motion Pack IMU that measures the vehicle angular rates and accelerations.
- A Global Positioning System (GPS) that gives, while surfaced, the vehicle position in order to correct any navigational errors accumulated during the submerged phases of a mission.

To observe the environment and potential contacts the vehicle uses:

- A Tritech ST725 scanning sonar or an ST1000 profiling sonar is used for obstacle avoidance and target acquisition. The sonar heads can scan through a 360° of rotation or a defined angle.
- A fixed focus wide-angle video camera is located in the nose and connected to a DVC recorder.

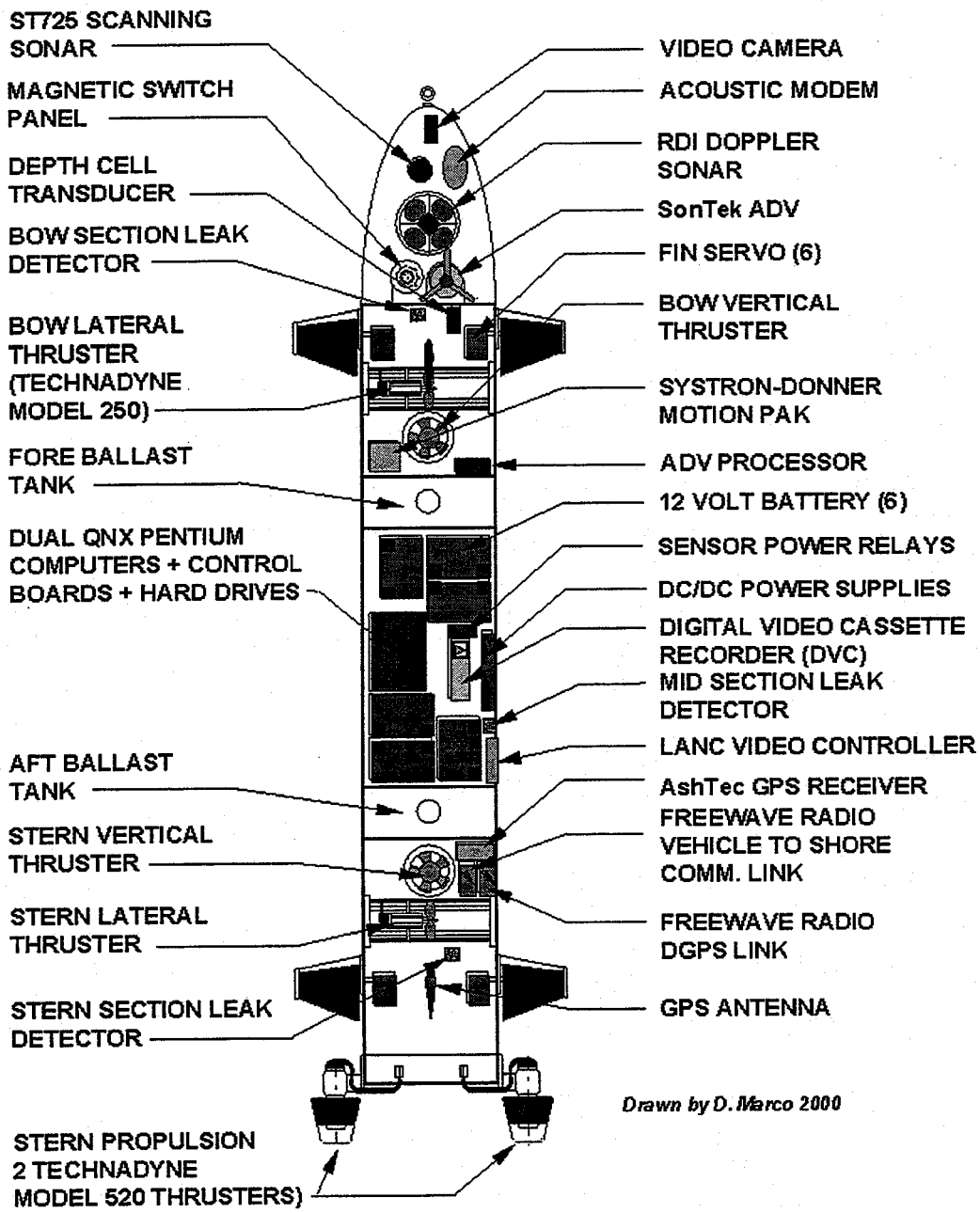


Figure III.2: Components inside the NPS ARIES AUV [Marco 2000]

To communicate with the surface operators and other vehicles, the AUV uses:

- Radio Modems for command, control and system monitoring while the vehicle is surfaced.
- While submerged, an acoustic modem is used for communications.

The vehicle is equipped with a dual-processor computer system supporting system software. It consists of two 166 MHz Pentium computers with 64 MB RAM, four serial ports, an Ethernet network adapter, and a 2.5 GB hard drive each.

2. Software

Both computers run the QNX real time operating system using synchronous socket sender and receiver network processes for data sharing between the two. All the software is written in the C programming language.

The ARIES AUV uses a tri-level software architecture called the Rational Behavior Model (RBM). RBM divides responsibilities into areas of open-ended strategic planning, soft-real-time tactical analysis, and hard real time execution level control. The RBM architecture has been created as a model of a manned submarine operational structure. The correspondence between the three levels and a submarine crew is shown is figure II.3.

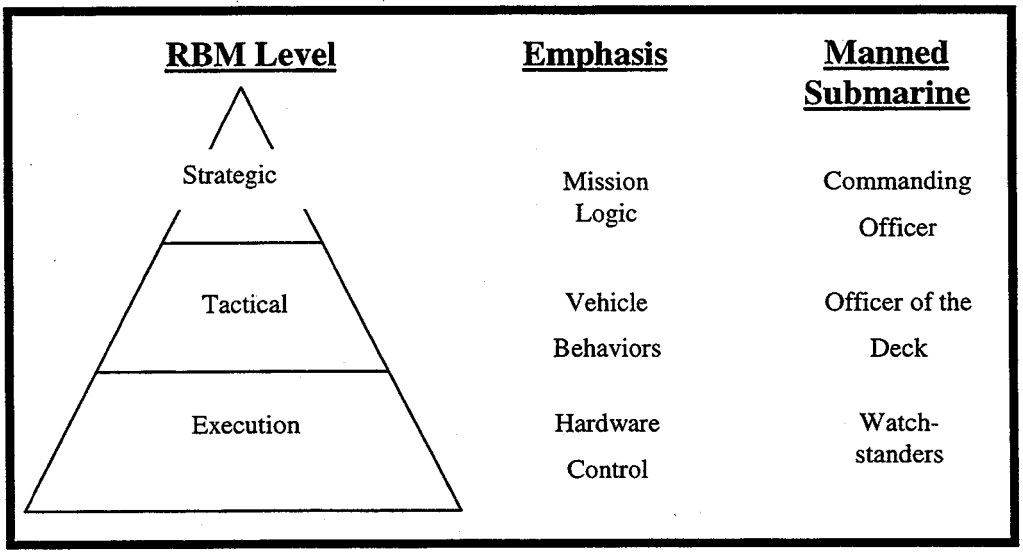


Figure III.3: The Rational Behavior Model tri-level architecture hierarchy with level emphasis and submarine equivalent listed [Brutzman 95]

The Execution level assures the interface between hardware and software. Its tasks are to underlay the stability of the vehicle, to control the individual device, and to provide data to the tactical level.

The Tactical level provides a software level that interfaces with both the Execution level and the Strategic level. Its responsibilities are to give to the strategic level indication of the vehicle state, completed tasks and execution level command.

The Strategic level controls the achievement of the mission goals. The mission specifications are inside this level.

C. SUMMARY

The ARIES AUV is a high technology underwater vehicle designed to operate in shallow water. Using a tri-level software architecture RBM model, it mimics a manned submarine operational structure. At the present time, the vehicle is fully operational and operates regularly in the Monterey bay. Only software enhancements are still required.

III. VIDEO SYSTEM DESIGN

A. INTRODUCTION

The complexity of an AUV and the hazardous underwater conditions introduce many requirements that must be considered when designing a new onboard system. Any new device, which will be integrated in the vehicle, has to match with the following characteristics:

- smallest size and lightest weight
- powered by 12 or 24V DC (power supply available on the vehicle)
- low power consumption in order to increase the robot endurance
- controllable by a computer with a convenient link (serial connection)
- for device installed out of the hull, it must have a 100m depth capability and remain resistant to saltwater corrosion
- easy maintenance
- price in accordance with the vehicle's one

The video system described in this chapter was designed following these points. This chapter provides a complete description of the system and each component device (previous, new and future), including details about actual implementation.

B. OVERALL DESCRIPTIONS

Figure III.1 shows a global view of the video system. It is composed of a camera connected to a Digital VideoCassette (DVC) recorder. The onboard computer is interfaced to the recorder via a logger card allowing remote control. An On-Screen Display module superimposes vehicle position data on the image. A video transmitter "links" the video system to an external receiver, such as an operator on board a support ship. This arrangement is straightforward and effective.

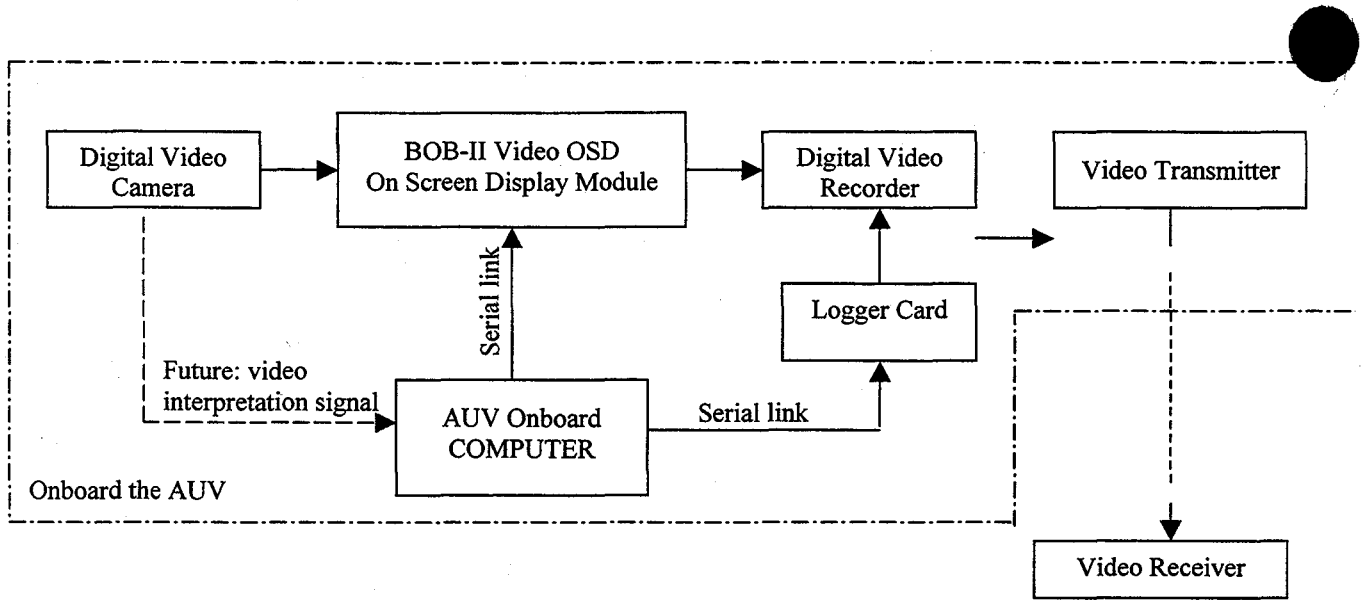


Figure III.1: Scheme of the global video system

C. DIGITAL VIDEO CAMERA

The video camera is the SST-1370 manufactured by [DeepSea Power & Light, San Diego, CA]. It combines video capture, with a ¼'' monochrome CCD, and a powerful lighting system, with thirty-five bright red Light Emitting Diodes (LED). Both are packed into a housing less than 3.5 cm (1.5 in.).

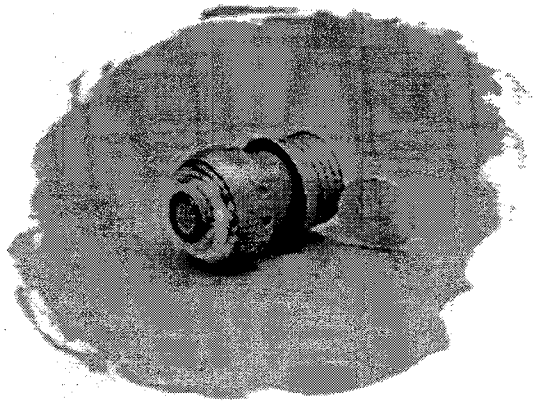


Figure III.2: Picture of the SST-1370 Camera [DeepSea Power & Light, 2000]

Durability and reliability are insured by a titanium housing and a sapphire crystal lens. The ¼'' lens is fixed focus, meaning there are no moving parts to break loose or wear. The SST-1370 is guaranteed in pressure to 750 meters (2.460 ft.) underwater and its video signal travels up to 600 meters (2000 ft.) over a cable. Its field of view in the water is of 76 degrees in horizontal and 67 degrees in vertical. On the AUV, the camera is situated at the extreme front of the nose and is looking down at the vertical.

D. DIGITAL VIDEO RECORDER

The video recorder is the GV-D300 Digital-Video Walkman manufactured by Sony. It has all the usual function of a video recorder hold in a small size (only 148 x 62 x 135 mm) and a light weight (970 g without the battery pack). Its average power consumption is 6.2W when recording and it uses 180 minutes long mini Digital Videotape. Lastly, but mostly, it is equipped with the LANC link allowing remote controls via a computer.

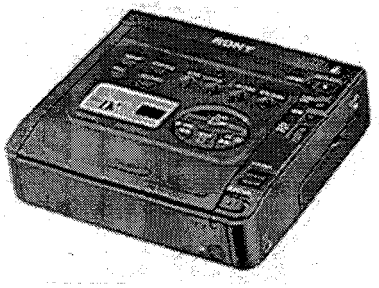


Figure III.3: Picture of the GV-D300 VCR [Sony, 2000]

E. LOGGER CARD (LANC PROTOCOL)

With a camera and a video recorder the video system is now operational but it needs to be automated and connected to the rest of the vehicle, i.e. interfaced with the onboard AUV computer.

1. Description

The Logger Card is the Model 199-100-014 manufactured by Sound Ocean System [Redmond, WA]. It is able to control a Sony video device equipped with the LANC link with a computer via a serial communication link. It requires a 24 V power supply.

2. LANC protocol

LANC, for Local Application Control, also called Control-L, is a bi-directional wired command protocol designed by Sony to control video products with an edit controller.

The controlling device is known as the Master and the controlled device as the Slave. At each field (rate of 50 Hz for PAL and 60 Hz for NTSC) a telegram is transmitted. Each one consists of eight words (bytes), transmits at a rate 9600 baud. Message takes approximately 10ms to transmit so therefore there is blank of 10 ms (PAL) or 6 ms (NTSC) between message.

Figure III.4: The LANC Telegram

Words 0, 1, 2, and 3 are send by the master device and used to send a command:

- Word 0 selects the transmission source (camera or commander as personal computer or editor) and the controlled device (camera or video tape recorder)
- Word 1 is used to select the command (Off, On, Record, Stop, Play, Pause, Rewind and Forward…) to send to the slave device.
- Words 2 and 3are used from a Tuner and Timer Unit to command a slave device.

Words 4, 5, 6 and 7 give information about the controlled device:

- Word 4 gives the slave status (what is doing and in which status it is).
- Word 5 gives the guide codes for words 6 and 7. This code is transmitted at least in every 4 fields.
- Words 6 and 7 give the value of the function (decimal counter, real-time counter...) selected by the word 5.

3. Control Code

The Logger Card is an interface between a computer and a Sony video device. It takes care of the timing requered by the LANC protocol and allows easily controlling of a video recorder via a serial link. The serial link operates a 9600 baud, 8 data bits, 1 stop bit with no parity and the LANC link uses a 2.5 mm sub-mini stereo jack.

All commands from the Logger Card are 12 bytes long. The strings start with the 'STX' character (byte 1), then have ten ASCII characters and end with 'ETX' (byte 12). Byte 2 selects the command mode ('RC' to use pre-recorded command strings and 'LC' to use LANC code command). Bytes 3 to 11 select the order to send to the video recorder. The card offers seven pre-recorded commands (Off, On, Record, Play, Stop, Pause and Rewind) corresponding at Byte 3 values ranging from 0 to 6. To use other commands, LANC code needs to be directly inserted: Bytes 1 and 2 corresponding to LANC word 0, and Bytes 3 and 4 corresponding to LANC word 1. A typical example: 'STX L C 1 8 3 8 0 0 0 0 ETX' is the string command to send the Fast Forward command.

The Logger Card also enables reading words 4 to 7 of the LANC message by sending the following command string (STX L I 0 0 0 0 0 0 0 0 0 ETX). LANC word 4 represents the status of the video recorder. This feedback allows checking the result of a prior command, or verifying the recorder status.

A C-language program called `LancControl.c` has been written to allow the AUV onboard computer to command the Video Recorder via the Logger Card. Only the primary useful functions (Off, On, Record, Play, Stop, Pause, Rewind and Forward) are included. After being wired and tested in the laboratory, the card has been installed in the vehicle and the `LancControl.c` program has been integrated into the AUV software.

The current Logger Card was purchased in mid 1999. New, smaller, lighter and cheaper Logger Cards are now available. Future work includes replacing this Logger Card with an Interface Adapter Model RS-4/L manufactured by [Addenda Electronics, Monterey, CA] which weighs only 60g (vice 500g), is five times smaller and doesn't require an external power supply.

F. THE ON-SCREEN DISPLAY

The video images produced by the camera do not provide enough information to be used efficiently because they do not indicate when and where they were recorded. An On-Screen-Display (OSD) adds text on a video signal.

1. Description

The On-Screen Display (OSD) is the BOB-II OSD manufactured by [Decade Engineering, Turner, OR]. It lets a PC or a microcontroller superimpose up to 308 characters over a standard NTSC or PAL video signal. Simple commands control its operation. It measures 3.5 x 1.05 x 0.35 inches and weights less than 0.5 oz. BOB-II needs to be powered between +8 and 16V.

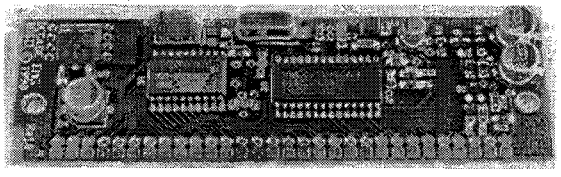


Figure III.5: Picture of the On-Screen Display [Decade Engineering, 2000]

It provides 128 different ASCII characters patterns of 12 x 18 pixels bitmaps, including upper and lower case, that may displayed, in 28 columns and 11 rows with specific character and background colors.

All displayed text characters and display configurations (position, color, blink...) are controlled via a serial communication link using a one-way data link with the following protocol: 9600 baud, 8 data bits, no parity, 1 stop bit and inverted data (0 = 5V and 1= 0V).

2. Inverted serial data interface

As BOB-II expects inverted serial data with 5V logic levels to be connected to the COM port of a computer (that is a 'true' RS-232 circuit), an inverting receive data interface (Figure III.6) was built and added between the computer and the on-screen display card.

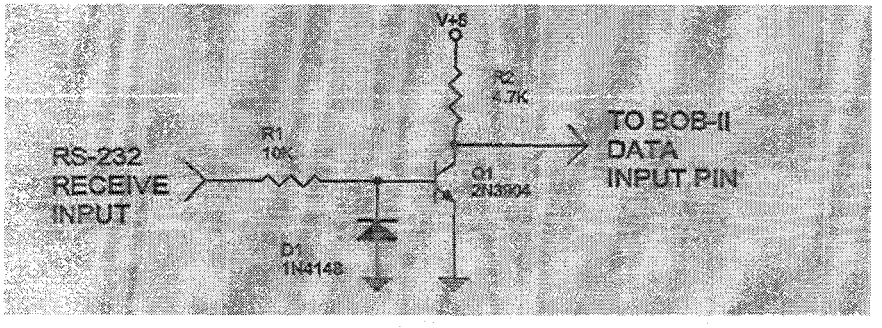


Figure III.6: Scheme of the inverted serial data interface

3. Which information is displayed?

The AUV computing environment contains great deal of data, so a choice of the most useful is necessary in order to keep most of the picture free of added characters. Thus the question of interest is: "Which information is needed in a mission's video?"

The primary data of interest is the position of the vehicle so that a location can be associated with each image. The x and y coordinates of the world coordinate system (x axis is directed trough the North and y axis through the West) give the vehicle position. Then the vehicle's depth and altitude are also displaying in meters. The date and the time (Greenwich time) allowing easy video archiving are the last two data displayed. All this data are displayed in S.I. units, giving an easy international understanding.

4. Control Program

A C-language program called `Bob.c` has been written to allow a real time display of all these selected data on the image provided by the camera. This program uses the specific BOB-II Control Protocol that employs its own syntax to send the commands. Communicated data include position of the cursor, clearing the screen, etc. The program loops every second

giving a high precision in the displayed information. After laboratory set up (wiring, housing) and testing, BOB-II was installed inside the vehicle and the program code integrated into the AUV software. Figure III.7 shows an image with the superimposed information.

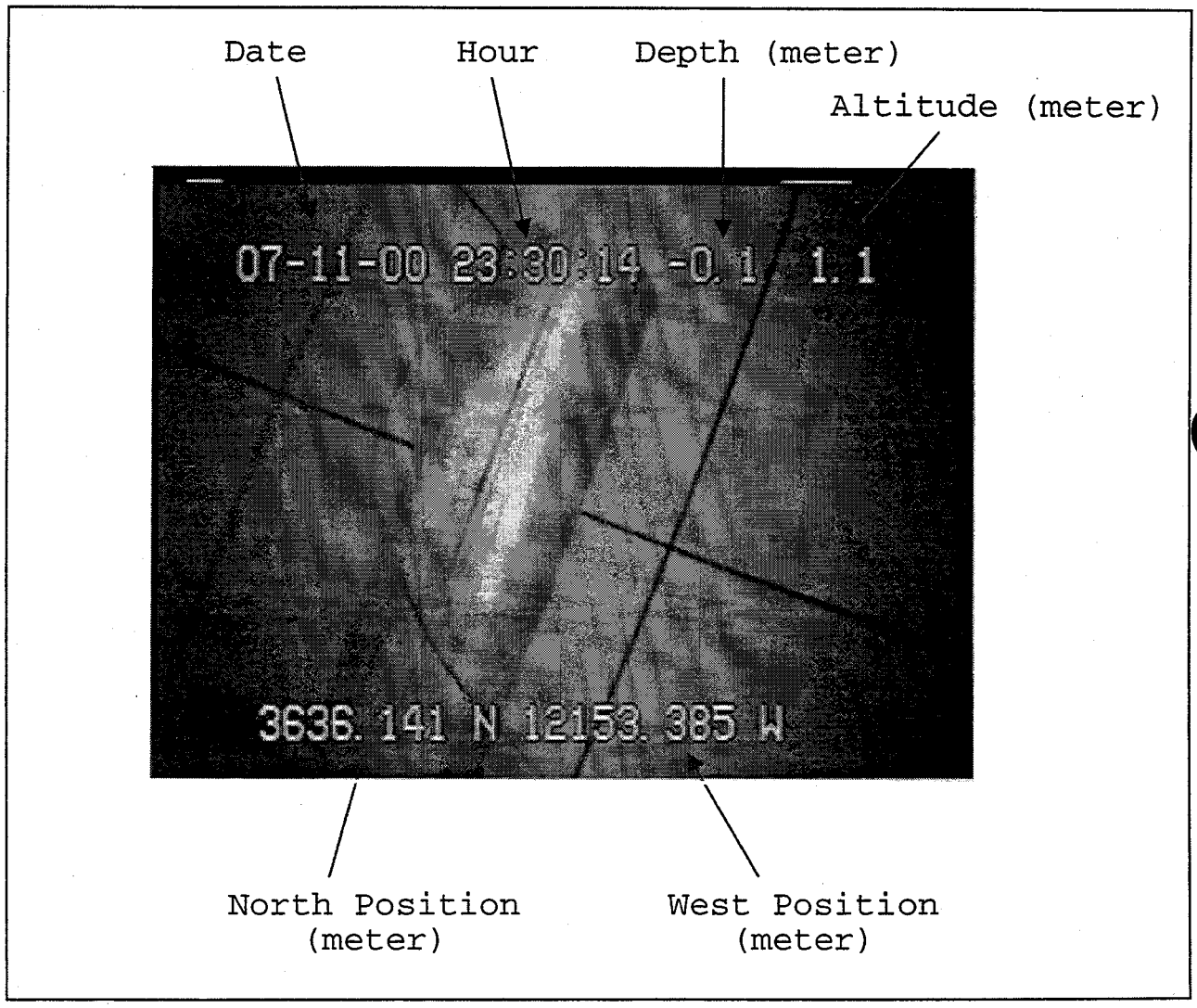


Figure III.7: AUV video image of a pipe, taken in the laboratory test tank

G. VIDEO TRANSMITTER

The video Transmitter and Receiver are two devices that permit accessing the video images without have the vehicle onshore. The important part is the transmitter, which is installed on the vehicle. The one selected is the Ultra Link Video Transmitter manufactured by [First Witness, Mt Sidney, VA]. This is a 2.4 GHz wireless video transmitter transmitting video over 15 miles line of sight. Powered by 12Vdc, it is very small (only 2.2 x 2 x 0.765 inch.) and doesn't need a high gain antenna. It transmits NTSC or PAL video signal using one channel.

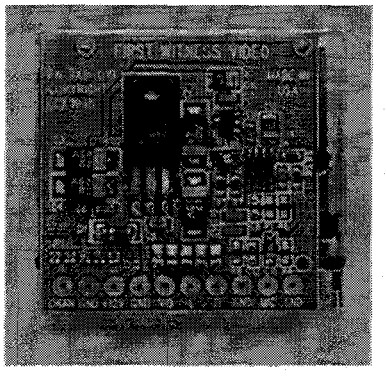


Figure III.8: Picture of the video transmitter [First Witness, 2000]

The transmitter implementation is currently under study. It has to be associated with a pre-amplifie and an antenna in order to get the best video-image quality whenever the vehicle can transmit. The antenna needs to be positioned on a non-submerged part of the hull when the vehicle is surfaced to transmit.

The Ultra Link Wireless Video Transmitter is connected to a receiver equipped with a 6db antenna. Figure III.9 provides a picture of the video receiver.

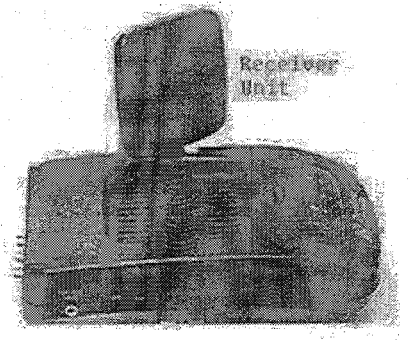


Figure III.9: Picture of the video receiver [First Witness, 2000]

This video transmitter is currently in use in Santa Cruz California to transmit video images from a camera situated on the beach, to a 1000 feet (330 meters) distant computer linked to Internet. The images transmitted, available at `http://www.santacruzweb.net/schook/schookcamns1.asp`, provide good quality.

H. EXPERIMENT

Several missions were performed in Monterey Bay in order to test the video system, recording underwater video images with the vehicle data superimposed. However, image quality is poor and some adjustments with the camera are needed. Figure III.8 shows an example image recorded with brightness problem. Add a color filter on the camera lens can be a solution to attenuate this problem.

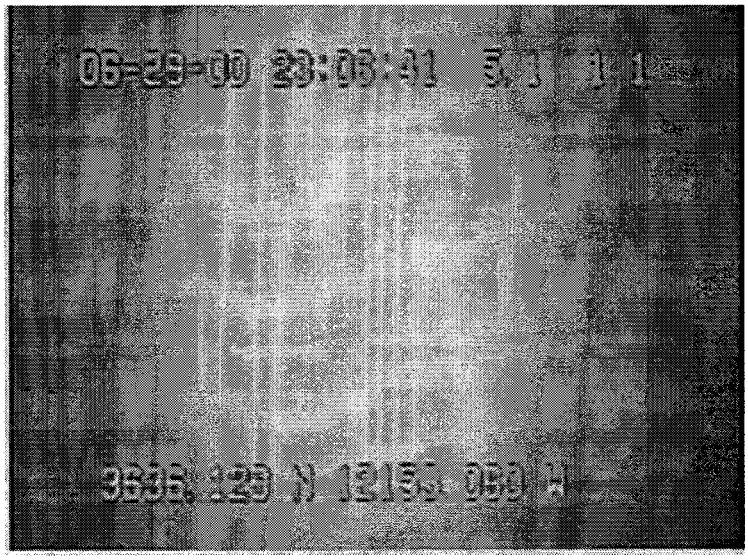


Figure III.10: Example of a recorded video image with brightness problem

I. SUMMARY AND RECOMMENDATIONS

The autonomous vehicle now has an internally controlled video system. The onboard computer could decide in function of a pre-defined position or a sonar information to record

images. Video images are easily archived since time and position information is superimposed.

Future addition of a video transmitter will permit remote monitoring of camera imagery for rapid reporting and retasking. Improvements have to be realized on the camera in order to record better quality video images. Signal connections for direct interpretation of video images by an AUV computer for navigation and contact classification is an important future hardware task.

IV. SIMULATION OF A VIDEO GUIDANCE SYSTEM

A. INTRODUCTION

This chapter describes validation and integration of the Optical Guidance System for the Recovery of an Unanimated Underwater Vehicle [Deltheil, Hospital, Leandri, Brutzman, 2000] in the NPS AUV.

Simulation is a low-cost and safe way to prepare new experimentation. Since the underwater environment is hazardous, remote and hostile, this system is evaluated using the AUV Virtual World, which is an extensive set of software simulating the behavior of the real AUV. This chapter provides a description of the visual tracking system, the simulation software and simulation results.

B. THE RECOVERY SYSTEM

1. Operational Context

The application of the optical guidance system is the recovery between an AUV and a host submarine. At the end of a mission, the AUVs need to be retrieved in order to collect the recorded data and to recharge the batteries. Thus after the vehicle has finished its mission, it has to proceed to a predetermined rendezvous area for recovery on board the submarine. The AUV needs a very accurate guidance system in order to steer to the recovery device installed on the submarine. For this reason, an additional guidance system coupled with the nominal navigation system may be a way to ensure safe vehicle control through the flow around the slowly moving submarine [Deltheil, Hospital, Leandri, Brutzman, 2000].

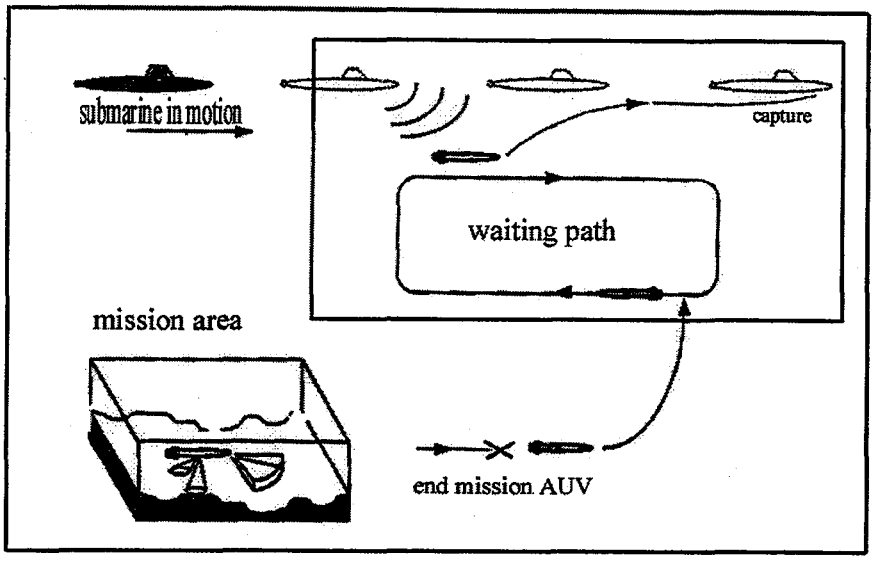


Figure IV.1: AUV recovery process [Deltheil 96]

2. Technology of the additional guidance system

The technology chosen for this guidance system is optical, using a light source tracking by a camera. A high-power light source is placed on the submarine and the AUV uses its camera to track the light source. Movement relative to the light source helps to determine an accurate heading. A simple control algorithm was developed which executes rapidly and robustly. This technology was selected due to its simplicity (compared to an image recognition system), its reliability and its rapidity of execution.

Onboard the AUV a vision module (processor) can analyze the images provided by the camera and deduce orders of motion to send to the navigation system. In our current implementation, this vision interpretation is performed in software.

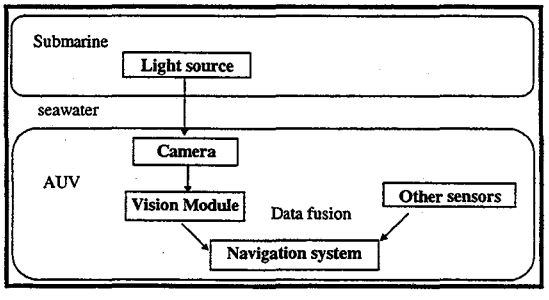


Figure IV.2: Functional diagram showing vision-based AUV tracking of a submarine beacon light [Deltheil 97]

3. Assumptions

This system has several assumptions for proper operations:

a. The ambient luminosity is sufficiently low and diffuse so that the luminous flow emitted by the light on the submarine could not be mixed up with other source light (sun light).
b. The AUV speed allows it to reach the submarine.
c. The AUV roll angle is negligible (almost zero).
d. When starting the recovery, the AUV position allows it to see the tracked light through its camera.

C. SOFTWARE SIMULATION

This section describes the software designed to simulate the video guidance system. It uses three sets of existing software: the AUV virtual world, the robot control software, and the image synthesizer Astyanax [Hospital 1996] that are each running on Windows-based personal computer. Previous simulations ran on SGI workstations.

1. The AUV Virtual World

The AUV Virtual World is software designed to reproduce real-world robot behavior with complete fidelity in the laboratory [Brutzman 94]. The NPS AUV software interactions with the virtual world occur in two programs: `execution` and `dynamics`.

`Execution` corresponds to the real vehicle navigation control software. In function of orders, it controls the vehicle by reading, updating and commanding the AUV's controlling hardware. This is a sense-decide-act, closed-loop control program that repeats every 100 ms in order to assure vehicle stability.

`Dynamics` simulates the underwater environment of the vehicle. It represents the underwater vehicle hydrodynamics model. It provides realistic sensor values in response to `execution` control orders.

When running the virtual world, from the orders of a mission script, `execution` calculates the orders to send to thrusters, propellers, rudders and planes. These orders are transmitted to `dynamics` that processes the static and dynamics parameters of the AUV.

2. The 3D Virtual World 'Viewer'

The virtual world is associated at a 3D Graphics viewer allowing seeing clearly how the AUV behaves in its environment. It uses the VRML language. This is only a visual interface and does not affect the performance of the other software. It also runs in real time (i.e. the same time as `execution` and `dynamics`), or can be operated in playback mode using the AUV telemetry values recorded during the mission. The PDU Player utility provided in the DIS-Java-VRML distribution can be used to record and playback such missions.

3. The image synthesizer software : Astyanax

Since the AUV Virtual World is used to simulate the vehicle behavior, what the vehicle sees through the camera needs also to be simulated. The Astyanax software is used to achieve this synthetic imaging task. It is based on the Pov-Ray algorithms and syntax [Pov-Ray version 3.0]. This software performs the simulation of a camera based on a ray-tracing system algorithm including the physical phenomena characteristics of the water (attenuation due to absorption and scattering).

With a description of the scene, composed of objects (geometry and position), a light source, the camera definition and the underwater medium, the image synthesizer generates images (set of pixels) of the scene from the viewpoint perspective of the underwater camera. The generated images are Bitmap (`.bmp`) format.

Figure IV.3: Image created by Astyanax

Ray-tracing is a slow rendering technique because it calculates an image of a scene by simulating all the way rays of light travel in the real world. Although this is not usually considered a real time algorithm, Astyanax gives satisfactory results in near real time.

4. Integration of the recovery system to the virtual world

The major modifications due to the integration appear in the `execution.c`, `parse_functions.c`, and `globals.c` files of `execution`.

The vehicle navigation control uses different control modes (HOVER, WAYPOINT, ROTATE...) described in `execution`. Each one corresponds into a specific way to move for the vehicle, i.e. to utilize the fins, rudders, thrusters and propellers. For the tracking light system, a new control mode, called FOLLOWLIGHTCONTROL was set up. When the keyword FOLLOWLIGHT is read in the `mission.script` file, the navigation control become FOLLOWLIGHTCONTROL and all the tracking process is running until the vehicle attains the light. Then the navigation followlight control mode stops, and `execution` shifts to the following command found in the `mission.script` file.

Each control mode is described in a specific function that is executed when its related control mode is activated. `Compute_followlight_control` is the function associated with the followlight control mode. This function presents all the recovery process and the vehicle navigation control allied. It is linked with the vision module that feed it with the motion orders.

During the followlight process, at each `execution` loop, the vision module runs Astyanax who processes the scene image viewed from the AUV camera, according to the Astyanax scene and the position of the vehicle. The vision module gets this image, calculates the consequent tracking orders and sends them to `execution`. Then the normal processing flow resumes: `execution` calculates the full set of orders to send to thrusters, propellers, rudders and planes. These orders are relayed to `dynamics`, instead of the actual actuators, so that `dynamics` can simulate actual hydrodynamic response.

The vision module functions, from which an image is calculated and the motion orders deduced can be found in the `Asty_static.cpp` and `Imgrytr.cpp` files.

D. SIMULATION SETUP

The computer used for observing the simulation must be configured for the Virtual World 'Viewer.' Installation instructions are available at the DIS-Java-VRML WebPages [`http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/`]

Three files are needed to run a mission:

a. The AUV mission in a `mision.script` file, including the keyword FOLLOWLIGHT, using the AUV coordinates systems.
b. The Astyanax scene in the `Astyanax.scn` file using the specific Astyanax scene description syntax. Description of the AUV environment (object, light, and seawater characteristics) must be defined in the Astyanax coordinates system.
c. The Viewer scene in VRML language. Description of the same scene (objects and light) that this one in the `Astyanax.scn` file in the Viewer coordinates system.

AUV execution, Astyanax and the Virtual World 'Viewer' use each a different coordinates system. The figure below shows the relations between the three coordinates systems.

$$X_{AUV} = X_{3D\ Viewer} = Z_{Astyanax}$$
$$Y_{AUV} = Z_{3D\ Viewer} = X_{Astyanax}$$
$$Z_{AUV} = -Y_{3D\ Viewer} = -Y_{Astyanax}$$

Figure IV.4: Relation between AUV execution, Astyanax and Viewer coordinates systems

Units are also different for each one. Astyanax uses (meters, degrees), the Virtual World 'Viewer' uses (meters, radians), and AUV execution requires (feet, degrees).

To launch the software:

a. Run `dynamics` in a DosWindow

```
~dynamics> java dynamics loop
```

b. Run `execution` in a DosWindow

```
~execution>execution remote [computername] mission
                followlight real-time
```

c. To see the mission with the viewer in real time, run the file `AuvInBeachTanksWithFollowlightSubmarine.wrl` in Netscape with DIS-Java-VRML installed. This file must include the VRML scene in meters. Choose then the dynamicsAUV viewpoint.

d. To see the mission with the viewer in playback (when the simulation is finished) the .wrl file needs to be created by running in a DosWindow the invocation:

```
vrtp\demo\auv> java demo.auv.TelemetryPlayback
                mission.output.telemetry >> fileName.wrl
```

The `mission.output.telemetry` file in the `vrtp\demo\auv>` directory must be this one created during the mission simulation.

Some modifications are needed in the `TelemetryPlayback.java` file. The VRML scene of interest needs to be integrated into for proper operations. Improving the usability of this program is an important area for future work.

E. VIEWING SIMULATION RESULTS

Several simulations were conducted in order to validate the software and test the recovery navigation control algorithm.

A scene made of a static submarine, a light and the optical properties of the seawater is described in the `Astyanax.scn` file. The same scene (submarine and light) is described in a VRML scene. A mission script including the FOLLOWLIGHT keyword is expressed in a `followlight.mission.script` file.

The simulated camera is positioned horizontally at the vehicle bow like the one currently implemented on the ARIES AUV. Simulation can use either color or gray camera. Both configurations provided the same results for the recovery, an important result.

After running a mission, the Bitmap files images can be showed either using a slide-show or compiling them into a video format `.avi` file creating a video clip. The `bmp2avi`

freeware, available at `http://www.homeusers.prestel.co.uk/cherryjam/utilities.html` was used to get the `.avi` file from the Bitmap images. To create a video clip, run the invocation `C:\auv\astyanax\images> bmp2avi -i Image -o mission.followlight -f 10 -k 1` with the `bmp2avi.exe` program in the `C:\auv\astyanax\images` directory.

The possibility of integrating the mission replay viewer and the mission video clip on the same screen offers of simultaneously watching the AUV and the AUV camera. This will be an important new capability to add in future versions of the virtual world.

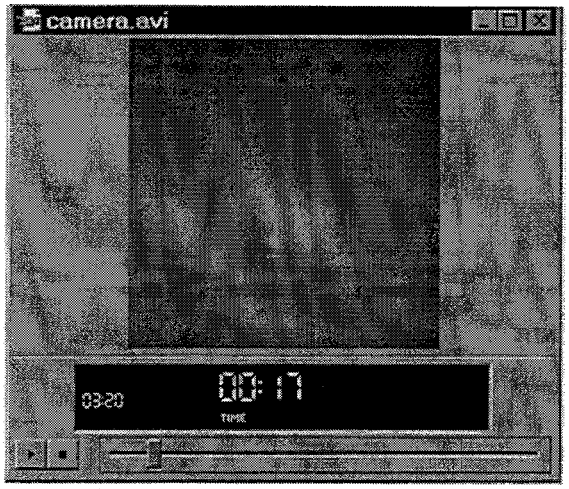


Figure IV.5: Recovery process Astyanax viewer, paying back a movie of all synthetic `.bmp` images stitched together in an `.avi` file

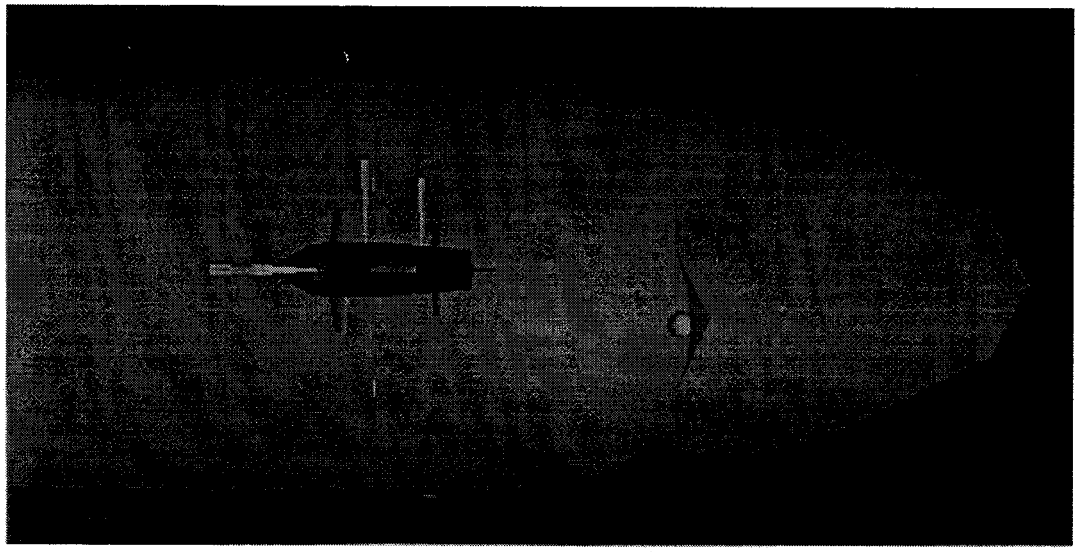


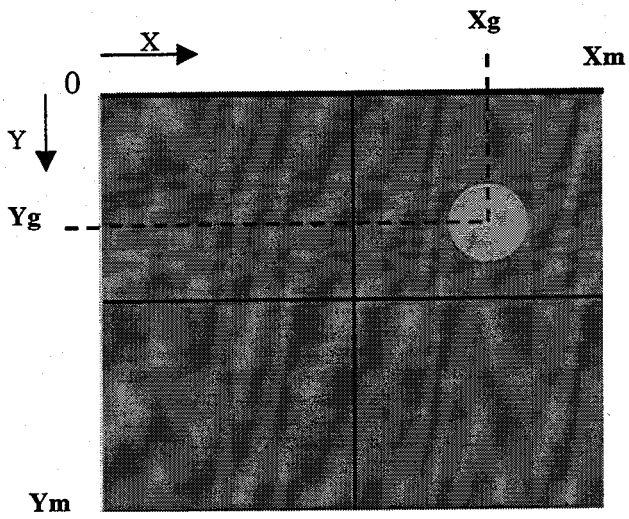
Figure IV.6: Recovery process 3D viewer: Netscape, Cosmoplayer and DIS-Java-VRML. View of scene `AuvInBeachTanksWithFollowlightSubmarine.wrl`

F. TRACKING ALGORITHM

This section describes how AUV control orders are deduced from synthetic (or real) video images

1. The followlight algorithm

The vision module control algorithm is designed to track towards the center of the camera screen with the light source.



Xm: max number of pixels in X
Ym: max number of pixels in Y

Xg and Yg: highest luminosity pixel position

Figure IV.7: View of the camera screen

After it has tracked the barycentric position of the highest luminosity area on the image, the vision module returns a commanded heading angle, a commanded depth and a commanded speed correcting the position error from the center of the camera. Figure IV.8 shows the followlight algorithm.

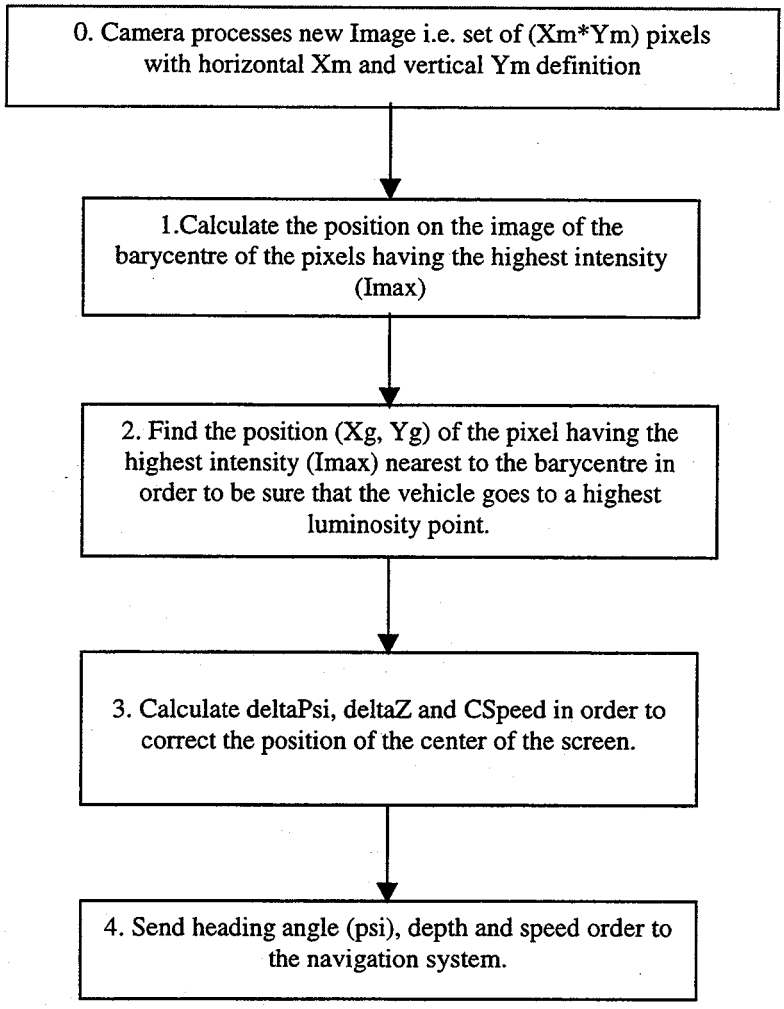


Figure IV.8: Followlight algorithm

The passage from the camera screen to the "real world" is doing by using laws allowing getting well-advised commands:

a. The depth order is the sum of the current depth (Z) and *deltaZ* (*deltaZ* represents the depth error).

$$deltaZ = ((2Yg - Ym)/Ym)$$

If the Yg value is less than 9/20 and or more than 11/20 of Ym, Yg is considered important, so *deltaZ* becomes:

$$deltaZ = 10 * ((2Yg - Ym)/Ym)$$

b. The heading order is the sum of the current heading (psi) and *deltaPsi* (*deltaPsi* represents the heading error).

$$deltaPsi = \arctan((2Xg - Xm)/Xm)$$

c. The relative speed order *CSpeed* is an offset-distance calculation. The *CSpeed* value goes from 0 to 1 depending of the position of the light on the screen. If the light coincide with the center of the screen, *CSpeed* equals 1.

$$CSpeed = 1/2(e^{-\frac{(Xg-Xm/2)^2}{400}} + e^{-\frac{(Yg-Ym/2)^2}{400}})$$

CSpeed is then used as a multiplying factor by the AUV speed controller.

d. The vision module calculates also a value *d* that represents the distance of the light from the center of the camera screen. Nearest is the light of the screen center nearest is *d* of zero.

$$d = \sqrt{\frac{(2*Xg-Xm)^2+(2*Yg-Ym)^2}{(Xm^2+Ym^2)}}$$

2. The recovery process

Simulated missions revealed that the control laws described above allow the vehicle to reach the light. However, to increase the efficiency of the recovery process, further improvements can be added to the process.

a. Vehicle Recovery Process

The overall recovery process is described by the set of steps shown in Figure IV.9.

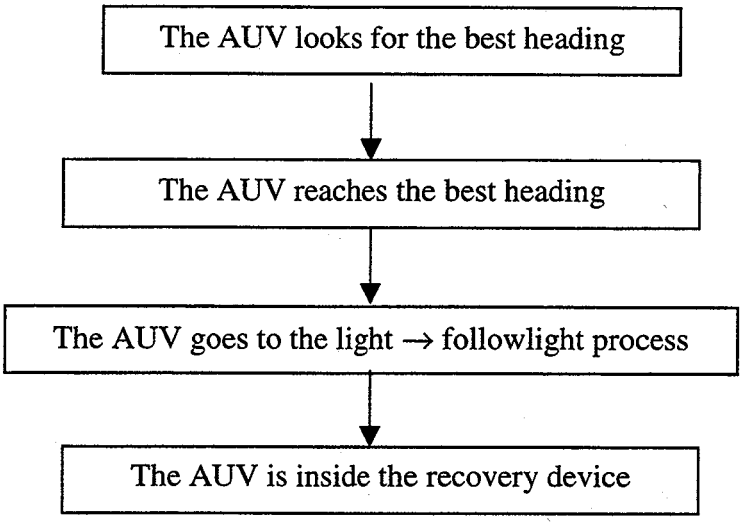


Figure IV.9: Vehicle recovery process

b. Starting: The First Image

A significant problem occurs if the first image provided to the vision module does not contain any light source. The vision module algorithm calculates a Xg and Yg coinciding with the center of the camera screen, so the depth and heading orders are null and the vehicle goes straight towards blackness rather than in the direction of the light. This behavior is undesirable.

To avoid this problem an initialization procedure is performed at the beginning of the recovery. When starting the process, the vehicle searches for the light by doing a complete 360° rotation. During the rotation, it updates the best heading value found for the highest light intensity. At the best heading, the camera field contains the highest intensity area nearest to the screen center (d minimal). After this procedure, the vehicle turns to the best heading found and then starts the followlight process. During this procedure, the vehicle keeps the same position using the HOVER navigation control.

c. Running: The Depth Command

For a normal depth command, the vehicle uses the lateral planes to reach the depth ordered. Thus the pitch angle varies. During the followlight progression, the pitch variations can introduce instability in the vertical position of the highest luminosity area on the camera screen. For example, one time deltaZ is negative so the vehicle goes up by

changing its pitch angle and also the inclination of the camera. The highest luminosity area position change quickly and the time after deltaZ becomes positive and the vehicle goes down... Thus opposite depth orders repeat continually. Figure IV.10 shows the evolution of Yg during the followlight process using the planes to reach a depth ordered.

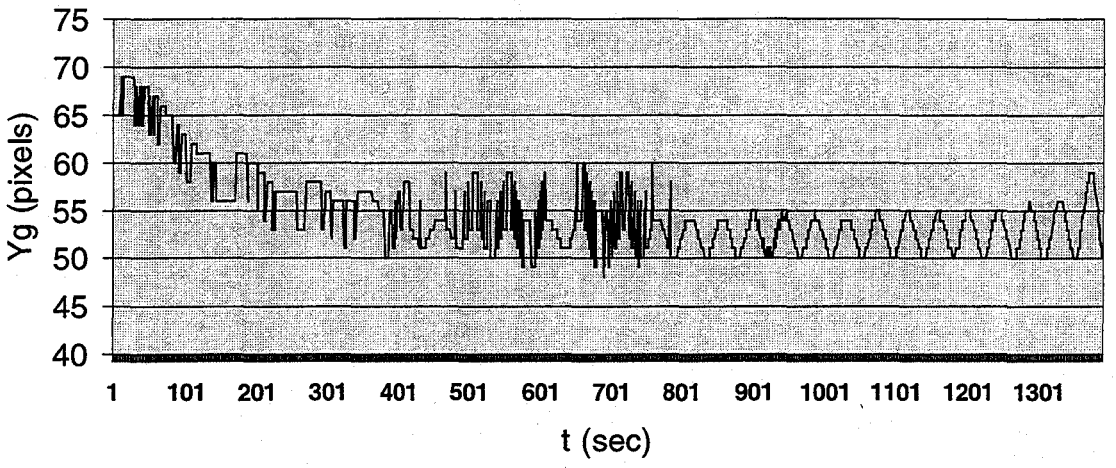


Figure IV.10: Yg evolution using the planes

To reduce this instability, when the vehicle is reaching the light, the depth command is executed only using the cross-body thrusters, and the planes are neutralized in order to maintain a zero pitch angle. This control mode effectively damps vertical pitch oscillations. The vehicle reaches then the light with stable motion controls. Figure IV.11 shows the Yg evolution with the planes neutralized.

Future work needs to explore the use of damped pitch-control coefficients for the lateral planes, in order to smoothly reach ordered depth with less expenditure of electrical energy while avoiding followlight algorithm instabilities. Cross-body thrusters needing much more power than the lateral planes.

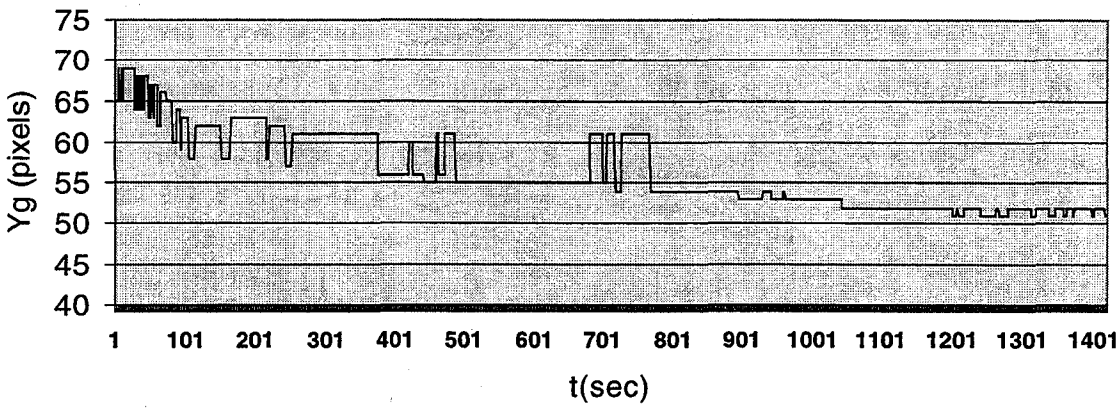


Figure IV.11: Yg evolution without using the planes

d. The End of the Recovery

The last task is the end of the recovery. The followlight progress has to be stopped when the vehicle is considered inside the recovery device installed on the submarine. To detect this position vehicle might use sonar or some other sensor.

During simulation, the recovery is considered complete when (a) the distance between the AUV and the light is less than three feet, and (b) the light is centered in the screen ($d < 0.06$), meaning the vehicle is directly pointing at the axis of the light.

G. SUMMARY AND RECOMMENDATIONS

The Virtual World has been updated to integrate the recovery system and the image synthesizer Astyanax. The run simulations demonstrate that the camera could be use as a navigation sensor and more exactly the feasibility of the visual guidance system using the couple camera-light. The followlight algorithm used allows the vehicle to reach the light and added improvements increase the process efficiency.

The followlight code now appears ready to be integrated, tested and calibrated using the actual video source in the vehicle. Different factors associated with the sonar, the speed commands, and the end of the recovery will require careful tuning.

Simulation is ordinarily insufficient to make strong conclusions and usually needs to be verified by experimental results. The next step for the simulation software might be to replace Astyanax with real underwater images in order to see the behavior of the vision

module with real images. Using a camera in a tank connected to the virtual world software is likely a good intermediate step.

Additional future work regarding the software will be to create an editor that will translate a X3D scene to both VRML and Astyanax.scn format. This editor might eliminate errors during the scene files writing.

The optical guidance system is used here for the recovery with a submarine but could also be with a recovery device tethered to a boat. The main requirement for this sensor is to be deep enough to avoid sunlight perturbation. Possibly this technique can be used for docking with buoys or underwater shelters at night.

VI CONCLUSIONS AND RECOMMENDATIONS

A. RESEARCH CONCLUSIONS

The goal of this work is to design an efficient video system, and also to demonstrate through simulation that the video can also be used as a sensor for the vehicle guidance. The video system also needs to efficiently perform the viewing, recording and interpreting the video images. This report successfully shows that the video system can become a major component on the ARIES AUV. The designed system achieves these tasks and has demonstrated satisfactory initial capabilities during missions in Monterey Bay. More work is needed on calibrating and optimizing signals received from the camera.

Simulation of a virtual recovery between the AUV and a submarine provides a good demonstration test regarding the use of video as a tracking sensor. The Virtual World has been updated to integrate the image synthesizer Astyanax and the recovery models. The followlight algorithm successfully enables the vehicle to track, follow and reach the beacon light. Algorithm improvements (best heading and depth command) added further efficiency to the recovery process.

B. RECOMMENDATIONS FOR FUTURE WORK

Some hardware capabilities have not yet been achieved in the construction of the video system. The poor quality of current video images needs to be corrected, with possible solutions including camera replacement, amplifier adjustment, lighting, blue-light filtering or other techniques. The radio transmitter for video signals needs to be installed on the vehicle. Both antenna and pre-amplifier will also need to be connected to the transmitter. Upgrading the Logger Card interface using new equipment [Addenda Electronics, Monterey, CA] can reduce electronics size and power consumption. Signal connections for direct interpretation of video images by an AUV computer are now needed, in order to evaluate the followlight tracking algorithm underwater. Further refinement and integration of navigation and contact classification will be important future developments.

Concerning the AUV recovery process, the followlight algorithm needs now to be tested and calibrated, using the submerged vehicle and real video. Such operations might be performed using optical beacons connected to moored buoys or submerged docking stations. Future work also needs to explore the use of damped pitch-control coefficients for the diving planes, tuned to the response of the followlight image-tracking function, in order to smoothly reach ordered depth with less expenditure of electrical energy. Such an improvement can also prevent the possibility of control instabilities when using the followlight algorithm.

Software work for the AUV Virtual World includes integration of synthetic video into postplayback or live 3D viewing, and representing the recovery scenes using the Extensible 3D (X3D) format for direct conversion into VRML and Astyanax/POVRAY formats. Recording of various simulation products for replay can also be made more usable.

APPENDIX A

This appendix provides:

- the BOB-II OSD Control Protocol
- the SST-1370 Camera Specifications

1. The BOB-II OSD Control Protocol [Decade Engineering, 2000]

DECADE ENGINEERING

5504 ValView Dr. SE, Turner, OR 97392-9517 (USA) ~ **tel:** 503.743.3194 ~ **fax:** 503.743.2095
email: decade@att.net ~ **web site:** http://www.decadenet.com

BOB-II Control Protocol V1.21 ~ May 2, 2000

Command	Description
{A	Clears the screen and sets the 'cursor' to top left home position (X=00, Y=00). Pause 5mS before resuming data transmission to BOB-II*.
{BE	Enables text display. This is the default (start-up) mode.
{BD	Disables text display without erasing display RAM. Characters may be written into display RAM while the display is disabled.
{Cxxyy	Sets 'cursor' to the indicated position. "xx" is the two-digit decimal ASCII column number (00~27) and "yy" is the row number (00~10). No internal range check.
{Dn	Character cell background color (local mode). "n" must be 0~7. Depends on {K commands.
{En	Character color for following characters transmitted (local mode). "n" must be 0~7.
{Fn	Screen color (local mode). Default color is blue. "n" must be 0~7.
{GE	Blink enable. Following characters flash in the display.
{GD	Blink disable. Following characters display without flashing.
{HN	Internal video level select (default). No external video level controls needed.
{HX	External video level select. See BOB-II pin description document for more information.
{In	Character outline color (local mode). "n" must be 0~7.
{JE	If {KE is used, just the written character cell backgrounds are colored according to background setting {Dn. This is the default mode.
{JD	Entire character display area is colored black (local mode). {En affects character foreground & background simultaneously. {Dn & {In are ineffective. {Fn controls screen border color.
{KE	Character background enable. Following characters show cell background color*.
{KD	Character background disable (default). Following characters display no cell background.
{MF	Local mode select. BOB-II generates video on-board. Pause 300uS before sending more data.
{MM	Automatic Genlock/Overlay (default) mode select. BOB-II superimposes text on video from external source. Allow 3 seconds for lockup. Monitor pin 15 if automatic mode switching is possible during data transmission. See BOB-II FAQ for more information.
{T...<ESC>	Literal byte values up to the escape character are written to display RAM without translation, making the non-ASCII characters available (see character set document).

* {KE followed by {A paints all character cells with background color.

Serial communication parameters are: **8N1** (8 data bits, no parity, 1 stop bit). Bit rates other than 9600bps are configured by connections to BOB-II module pins 12~13 at installation time. See BOB-II Pin Description document for details.

Commands sent to BOB-II must be prefixed by the left curly brace character: { All commands except {T employ a fixed-length format, and do not require a command suffix. Multiple commands require a { prefix to each command in the string.

2. SST-1370 Camera Specifications [Deepsea Power & Light, 2000]

SST-1370 Combined Video & Lighting System

Specifications

Video		Electrical	
Image Sensor	¼" CCD image sensor	**Power**	8.7 to 13.2 volts DC
Number of Pixels	512 (H) x 492 (V)	**Current**	110 mA (camera), ~290 mA with LEDs
Resolution (TV lines)	380 (H) x 350 (V)	**Connector**	740-055-004
Lens	2 mm fl, f3.5	**Environmental**	
Focus	Fixed focus, factory set	**Operating Temperature**	-10 C to 50 C (14 F to 122 F)
Min. Depth of Field	½-inch	**Depth Rating**	750 meters (2,460 ft)
Scene Illumination	0.5 lux	**Weight air/water**	85 g/70 g (3.0 oz/2.5 oz)
Field of View in Air	111 (H) x 95 (V) x 122 (D) deg.	**Mechanical**	
Field of View in Water	76 (H) x 67 (V) x 82 (D) deg.	**Camera Diameter**	3.48 cm (1.37 in)
Lighting	35 red LEDs with variable intensity	**Camera Length**	5.00 cm (2.00 in), excluding collar (included)
Video Output	1.0 volt, peak-to-peak, at 75 ohm	**Housing Materials**	Sapphire crystal lens port. Impact-resistant LED window. Titanium housing.
Video Format	Monochrome; EIA or CCIR		
Shutter	Electronic, automatic		

Cables

Camera Connector	Cable Type	Length	Dry-End Termination
740-055-004: Also available as un-terminated whip.	RG-59 Triax **(TRI)**: 75 ohm video cable with polyethylene jacket.	**0-1000 feet**: Price is per-foot. Contact DeepSea for pricing.	**TOP/SS**: Standard. Also available on unterminated 18" whip.
	Kevlar Triax **(KTRI)**: 75 ohm video cable with internal kevlar braid and polyurethane jacket.		**Other**: Please contact DeepSea with requirements.

How to Order: 740-055-004) – ( ) – ( ) – ( )

Power & Viewing

SST-1370 heads are ideally suited to work as a system with our family of controllers, which include the ability to vary the LED light intensity, view the image with high resolution monitors, output the video signal to other monitors and VCRs, or make video recordings with *Hands-Free* audio overlay.

Power+	**(P+)** 120 vac/60 Hz (or 220 vac/50 Hz) to ~12 vdc power supply. Includes direct connection for SS3ILMP terminated cables. RCA video output interfaces with customer-supplied monitors and VCRs. Variable LED intensity controller allows fine adjustment of lighting for different conditions. A Ground Fault Circuit Interrupt (GFCI) is included on the AC cord for added safety.
Monitor+	**(M+)** A P+ with high-resolution surveillance monitor (>850 TV lines) enclosed in a rugged, aluminum case. A metal sunshade folds down to protect the CRT screen and front panel controls during transport. A tilt-stand improves viewing angle when system is on the ground. Includes direct connection for TOP/SS-terminated cables. Variable LED intensity controller allows fine adjustment of lighting for different conditions. Video in/out jacks allow hook up to external devices such as VCRs, camcorders and other remote monitors. A Ground Fault Circuit Interrupt (GFCI) is included on the AC cord for added safety.
Monitor+VCR	**(M+VCR)** (Release pending) An M+ with added VHS VCR and *Hands-Free* audio recording. Video/audio out jacks, CB mike input jacks, front-panel wireless microphone and playback speaker, one-button monitor/record/play.

Specifications subject to change without notice

DEEPSEA POWER & LIGHT 3855 Ruffin Rd. San Diego, CA 92123 USA TEL (858) 576-1261 FAX (858) 576-0219
Web: http://www.deepsea.com e-mail: info@deepsea.com

Rev. 12/2000

APPENDIX B

This appendix provides the C-code enable to command the the Logger Card (LANC) and the On-Screen-Display BOB-II

1. **`LancControl.c`**
2. **`Bob.c`**

1. LancControl.c

```
/*************************************************************************
*
   File: LancControl.c

   Purpose: This code edit commands to the Sony VCR via the Model 199-
            100-014 Logger Card, manufactured by Sound Ocean System
            through a RS-232 serial port.

*************************************************************************
**

#include <stdio.h>   /* Standard input/output definitions */
#include <math.h>
#include <fcntl.h>   /* File control definitions */
#include <string.h>  /* String function definitions */
#include <termios.h> /* POSIX terminal control definitions */

/*command strings

   Recorder Off:     STX R C 0 0 0 0 0 0 0 0 ETX
   Recorder On:      STX R C 1 0 0 0 0 0 0 0 ETX
   Recorder Record:  STX R C 2 0 0 0 0 0 0 0 ETX
   Recorder Play:    STX R C 3 0 0 0 0 0 0 0 ETX
   Recorder Stop:    STX R C 4 0 0 0 0 0 0 0 ETX
   Recorder Pause:   STX R C 5 0 0 0 0 0 0 0 ETX
   Recorder Rewind:  STX R C 6 0 0 0 0 0 0 0 ETX
   Recorder Forward:STX L C 1 8 3 8 0 0 0 0 ETX
*/

#define STX 0x02
#define ETX 0x03

main()
{
   /*declaration*/
   int i,j,k,n,fid,Code;
   char x_buf[1],c[1];
   char SendString[11];

   /*open serial port*/
   fid = open("/dev/ser6", O_RDWR | O_NOCTTY); /* This BLOCKS */

   if (fid == -1)
   {
     printf("open_port: Unable to open \n");
     exit(0);
   }

   /*nitialize string*/
   SendString[0]  = STX;
   SendString[1]  = 'R';
   SendString[2]  = 'C';
   SendString[4]  = '0';
```

```
SendString[5]  = '0';
SendString[6]  = '0';
SendString[7]  = '0';
SendString[8]  = '0';
SendString[9]  = '0';
SendString[10] = '0';
SendString[11] = ETX;


printf("What do you want to do?\n");
printf("Input the number for the action, 0-7 Only\n");
printf("\n");
printf("0. Recorder Off\n");
printf("1. Recorder On\n");
printf("2. Recorder Record\n");
printf("3. Recorder Play\n");
printf("4. Recorder Stop\n");
printf("5. Recorder Pause\n");
printf("6. Recorder Rewind\n");
printf("7. Recorder Forward\n");

scanf("%d",&Code);

switch(Code)
{
   case 0:
      SendString[3]  = '0';
      break;

   case 1:
      SendString[3]  = '1';
      break;

   case 2:
      SendString[3]  = '2';
      break;

   case 3:
      SendString[3]  = '3';
      break;

   case 4:
      SendString[3]  = '4';
      break;

   case 5:
      SendString[3]  = '5';
      break;

   case 6:
      SendString[3]  = '6';
      break;

   case 7:
       SendString[1]  = 'L';
       SendString[2]  = 'C';
       SendString[4]  = '1';
```

```
            SendString[5]  = '8';
            SendString[6]  = '3';
            SendString[7]  = '8';

         default:
            printf("Invalid Code\n");
            exit(0);
      }

      /*send string*/
      for(i=0;i<12;++i)
      {
         c[0] = SendString[i];
         printf("0x%x\n",c[0]);
         n = write(fid,c,1);
      }
      n = read(fid,x_buf,1);
      printf("0x%x\n",x_buf[0]);
      close(fid);
   }
```

2. Bob.c

```
/*************************************************************************
******************
      File: Bob.c
      Purpose: This code send to the On-Screen-Display BOBII the data to
            display on the video signal trough a RS- 232 serial port.
*************************************************************************
*****************/

#include "Bob.h"
#include <i86.h> /* For delay Function */

   double pi     = 3.14159265358979;
   double RadDeg = 57.2957795;
   double DegRad = 0.01745329;

   fsleep(Seconds)
   float Seconds;
{
   delay( (int) Seconds*1000.0);
}

main()
{
   int    Bob_Id     = 0;
   int    Month      = 0;
   int    Day        = 0;
   int    Year       = 2000;
   int    Hour       = 0;
   int    Minute     = 0;
   double Second     = 0.0;
   double X          = 0.0;
   double Y          = 0.0;
   double Depth      = 0.0;
   double Alt        = 0.0;
   int    GPS_Signal = 0;
```

```
double LatDeg0    = 0.0;
double LongDeg0   = 0.0;
int    Bob_Error  = 0;

char TimeString[27];
char GPSString[22];
char Mon[2],Dy[2],Yr[2],Hr[2],Min[2],Sec[2];
char LatString[12],LongString[13],Dep[5],Al[5];
char LatDir[1],LongDir[1];
char TS[30];
int SecInt;
int Year2p;

int Bob_Kill = FALSE;
pid_t LoopTimerProxy;
timer_t LoopTimerId;
struct itimerspec LoopTimer;
struct sigevent event;

int Hz;

int i,j,k,n,fid;
char ClrScn[2] = "{A";
char StartPos1[6] = "{C0101";
char StartPos2[6] = "{C0110";

int INIT_ERROR = FALSE;
int Bob_Shmid,BobFlag_Shmid;

double Coef1;
double Lat,Long;        /* dddmm.mmmmmm */
double LatDeg,LongDeg; /* ddd.dddddd   */
double LatD,LongD;      /* ddd.000000   */

/*Coef1 = 3443.9*(1852.0)*(pi/180.0);*/
Coef1 = 111318.8938906694;

Hz = 1;

if((Bob_Shmid = OpenBobShm()) == -1)
{
   printf("Cannot Attach Bob Shared Memory\n");
   INIT_ERROR = TRUE;
}

if((BobFlag_Shmid = OpenBobFlagShm()) == -1)
{
   printf("Cannot Attach Bob Flag Shared Memory\n");
   INIT_ERROR = TRUE;
}

fid = open("/dev/ser5", O_RDWR | O_NOCTTY); /* This BLOCKS */

if (fid == -1)
{
  printf("open_port: Unable to open \n");
  exit(0);
}

/* Get a Proxy for the Timer to Kick */
LoopTimerProxy = qnx_proxy_attach( 0, 0, 0, -1 );
if(LoopTimerProxy == -1)
{
   printf( "Unable to Attach Proxy." );
```

```
   return;
}

/* Attach to the Timer */
event.sigev_signo = -LoopTimerProxy;
LoopTimerId = timer_create(CLOCK_REALTIME,&event);
if(LoopTimerId == -1)
{
   printf( "Unable to Attach Timer." );
   return;
}

ResetBobFlagShm(BobFlag_Shmid);

/*
 * 1 nano-seconds before initial firing,
 * 1.0/Hz second repetitive timer afterwards.
 */
LoopTimer.it_value.tv_sec     = 0L;
LoopTimer.it_value.tv_nsec    = 1L;
LoopTimer.it_interval.tv_sec  = 0L;
/* Convert Hz into NanoSecond Period */
LoopTimer.it_interval.tv_nsec = (int) (1.0/((float) Hz)*pow(10.0,9.0));
timer_settime(LoopTimerId,0,&LoopTimer,NULL);

i=0;

while(TRUE)
{
   /* Wait for the Proxy */
   Receive(LoopTimerProxy,0,0);

   ReadBobShm(Bob_Shmid,&Bob_Id,&Month,&Day,&Year,&Hour,&Minute,&Second,

         &X,&Y,&Depth,&Alt,&GPS_Signal,&LatDeg0,&LongDeg0,&Bob_Error);

   if(Month<10)
   {
      sprintf(Mon,"0%d",Month);
   }
   else
   {
      sprintf(Mon,"%d",Month);
   }

   TimeString[0] = Mon[0];
   TimeString[1] = Mon[1];
   TimeString[2] = '-';

   if(Day<10)
   {
      sprintf(Dy,"0%d",Day);
   }
   else
   {
      sprintf(Dy,"%d",Day);
   }


   TimeString[3] = Dy[0];
   TimeString[4] = Dy[1];
   TimeString[5] = '-';

   Year2p = Year - 2000;
```

```
if(Year2p<10)
{
   sprintf(Yr,"0%d",Year2p);
}
else
{
   sprintf(Yr,"%d",Year2p);
}

TimeString[6] = Yr[0];
TimeString[7] = Yr[1];

if(Hour<10)
{
   sprintf(Hr,"0%d",Hour);
}
else
{
   sprintf(Hr,"%d",Hour);
}

TimeString[8]  = ' ';
TimeString[9]  = Hr[0];
TimeString[10] = Hr[1];
TimeString[11] = ':';

if(Minute<10)
{
   sprintf(Min,"0%d",Minute);
}
else
{
   sprintf(Min,"%d",Minute);
}

TimeString[12] = Min[0];
TimeString[13] = Min[1];
TimeString[14] = ':';

SecInt = (int) Second;

if(SecInt<10)
{
   sprintf(Sec,"0%d",SecInt);
}
else
{
   sprintf(Sec,"%d",SecInt);
}

TimeString[15] = Sec[0];
TimeString[16] = Sec[1];
TimeString[17] = ' ';

sprintf(Dep,"%4.1f",Depth);
sprintf(Al,"%4.1f",Alt);

TimeString[18]  = Dep[0];
TimeString[19]  = Dep[1];
TimeString[20]  = Dep[2];
TimeString[21]  = Dep[3];
TimeString[22]  = ' ';
TimeString[23]  = Al[0];
```

```
TimeString[24]  = Al[1];
TimeString[25]  = Al[2];
TimeString[26]  = Al[3];

TimeString[27] = NULL;

TS[0] = ' ';
TS[1] = ' ';
TS[2] = ' ';

for(i=3;i<30;++i)
{
   TS[i] = TimeString[i-3];
}

LatDeg  = X/Coef1 + LatDeg0;
LongDeg = Y/(Coef1*cos(DegRad*LatDeg)) + LongDeg0;

/* Isolate # of Degrees */
LatD  = (double) ((int) LatDeg);
LongD = (double) ((int) LongDeg);

/* Convert to dddmm.mmmmmm */
Lat  = LatD*100.0  + (LatDeg  - LatD)*60.0;
Long = LongD*100.0 + (LongDeg - LongD)*60.0;

if(Lat  > 0.0) LatDir[0]  = 'N';
if(Lat  < 0.0)
{
   LatDir[0]  = 'S';
   Lat = -Lat;
}

if(Long > 0.0) LongDir[0] = 'E';
if(Long < 0.0)
{
   LongDir[0] = 'W';
   Long = -Long;
}

sprintf(LatString,"%8.3f",Lat);
sprintf(LongString,"%9.3f",Long);
GPSString[0]  = LatString[0];
GPSString[1]  = LatString[1];
GPSString[2]  = LatString[2];
GPSString[3]  = LatString[3];
GPSString[4]  = LatString[4];
GPSString[5]  = LatString[5];
GPSString[6]  = LatString[6];
GPSString[7]  = LatString[7];
GPSString[8] = ' ';
GPSString[9] = LatDir[0];
GPSString[10] = ' ';
GPSString[11] = LongString[0];
GPSString[12] = LongString[1];
GPSString[13] = LongString[2];
GPSString[14] = LongString[3];
GPSString[15] = LongString[4];
GPSString[16] = LongString[5];
GPSString[17] = LongString[6];
GPSString[18] = LongString[7];
GPSString[19] = LongString[8];
GPSString[20] = ' ';
GPSString[21] = LongDir[0];
```

```
        GPSString[22] = NULL;

        /* Clear the Screen an send the Curser Home */
        n = write(fid,ClrScn,2);
        fsleep(0.01);
        /*n = write(fid,StartPos1,6);
        n = write(fid,TimeString,27);*/

        n = write(fid,TS,30);

        n = write(fid,StartPos2,6);
        n = write(fid,GPSString,22);

        if(Bob_Id>65534)
        {
           Bob_Id = 0;
        }
        ++Bob_Id;

        ReadBobFlagShm(BobFlag_Shmid,&Bob_Kill);
        if(Bob_Kill) break;
    }

    /* Clear Pending Proxies */
    while(Creceive(LoopTimerProxy,0,0) == LoopTimerProxy);

    /* Get Rid of the Timer */
    timer_delete(LoopTimerId);
    CloseBobShm(Bob_Shmid);
    CloseBobFlagShm(BobFlag_Shmid);
    close(fid);
}
```

APPENDIX C: Recovery Process Codes

This appendix provides the codes for the integration of the recovery process in the virtual world.

1. `global.h`
2. `global.c`
3. `parse_functions.c`
4. `execution.c`
5. `asty_static.cpp`

1. global.h

Code added to `global.h`

```
/*************************************************************************/
extern int    FOLLOWLIGHTCONTROL     ; /*  1=followlightin progress  */

/*************************************************************************/
extern double Light_Source_center_X; /*coordinates of the center of the light
     source in the astyanax coordinates system to calculate distanceAuvLight*/
extern double Light_Source_center_Y;
extern double Light_Source_center_Z;
extern double distanceAuvLight;/*distance between the center of the Astyanax
                                                    light and the AUV*/

/*************************************************************************/
```

2. global.c

Code added to `global.c`

```
/*************************************************************************/

int    FOLLOWLIGHTCONTROL     = 0;    /*  1=followlight in progress */

/*************************************************************************/

double Light_Source_center_X = 0;/*coordinates of the center of the light
                            source in the astyanax coordinate system*/
double Light_Source_center_Y = 0;
double Light_Source_center_Z = 0;
double distanceAuvLight=0;/*distance between the center of the Astyanax
                                      light and the AUV*/
/*************************************************************************/
```

3. parse_functions.c

Code added to `parse_functions.c`

```
/*************************************************************************/

else if ((strcmp (keyword, "FOLLOWLIGHT") == 0))
        {
      if (DISPLAYSCREEN) printf ("\n\n[Light-following starts]\n");
      if (DISPLAYSCREEN) printf ("\n\n[Look-Around starts]\n");
          FOLLOWLIGHTCONTROL = TRUE;
          HOVERCONTROL = FALSE;
        }

/*************************************************************************/

if ((HOVERCONTROL)||(WAYPOINTCONTROL)||(REPORTSTABLE)||(FOLLOWLIGHTCONTROL))
```

```
/*************************************************************************/
```

4. execution.c

Code added to execution.c

```
/*************************************************************************/

void            compute_followlight_controls            ();

/*************************************************************************/

else if (FOLLOWLIGHTCONTROL) compute_followlight_controls();

/*************************************************************************/

     if ((NOSCRIPT == FALSE) &&
         ((HOVERCONTROL)     || (TARGETCONTROL) ||
          (WAYPOINTCONTROL)  || (ROTATECONTROL) ||
          (THRUSTERCONTROL)  || (FOLLOWLIGHTCONTROL)))

/*************************************************************************/

void compute_followlight_controls ()
{
      static int    LOOK_AROUND_IN_PROGRESS = TRUE;
      static int    firstLoopInitialized = FALSE;
      static double previous_psi =0;
      static double psiLightReference =0;
      static int    ZeroLookAroundFlag = FALSE;
      static double psiStartLookAround = 0;
      static double xLookAround = 0;
      static double yLookAround = 0;
      static double zLookAround = 0;
      static int    HEADING_REFERENCE = FALSE;
      static double feet_per_meter = 3.28084; /* number of feet per
meter*/

      if ((TRACE || TRUE) && DISPLAYSCREEN)
            printf ("\n[begin compute_followlight_controls]\n");

      if (firstLoopInitialized == FALSE)
      {
            previous_psi = psi;
            psiLightReference = psi;
            psiStartLookAround = psi;
            firstLoopInitialized = TRUE;
            xLookAround = x;
            yLookAround = y;
            zLookAround = z;
      }

      if (LOOK_AROUND_IN_PROGRESS)
      {
            if (DISPLAYSCREEN) printf ("\n[LOOK-AROUND]\n");
```

```
        DEADSTICKPLANES = TRUE;
        rotate_command = 12.0;
        compute_rotate_controls ();
        lateral_command = 0.0;
        x_command = xLookAround;
        y_command = yLookAround;
        z_command = zLookAround;

        if (DISPLAYSCREEN) printf ("\n[Ray-Tracing]\n");
        RenderCalculation (x, y, z, psi, theta, phi);
        psiLightReference = CalculatePsiLightReference();

        if ((fabs(previous_psi-psi))>180.0) /*test of passage at zero of
                                            psi*/
              ZeroLookAroundFlag=TRUE;
        if ((ZeroLookAroundFlag==TRUE)&&(psi>psiStartLookAround))
           LOOK_AROUND_IN_PROGRESS = FALSE;

        previous_psi = psi;
 }

 if ((!LOOK_AROUND_IN_PROGRESS) && (HEADING_REFERENCE == FALSE))
 {
        if (DISPLAYSCREEN) printf ("\n[HEADING REFERENCE]\n");
        REPORTSTABLE        = TRUE;
        time_int_control_on  = t + 10.0; /* give PD control 10 seconds */
        ST725SCANMODE       = SONARSCANSWATH;      /* Forward Scan */
        DEADSTICKRUDDER     = TRUE;
        DEATH_SPIRAL_RESET = TRUE;
        rudder_command      = 0.0;
        x_command = xLookAround;
        y_command = yLookAround;
        z_command = zLookAround;
        psi_command = normalize(normalize2(psiLightReference));
        psi_command_hover = psi_command;
        compute_hover_controls ();

        if (DISPLAYSCREEN) printf ("\n[Ray-Tracing]\n");
        RenderCalculation (x, y, z, psi, theta, phi);
        SaveImage();

        if ((floor(10*psi)) == (floor(10*psiLightReference)))
              HEADING_REFERENCE=TRUE; /*vehicle is at its heading of
                                        reference*/
 }

if ((!LOOK_AROUND_IN_PROGRESS) && (HEADING_REFERENCE == TRUE))
 {
        static double CSpeed, DeltaPsi, DeltaZ;

        if (DISPLAYSCREEN) printf("\n[Light-Following]\n");

        if (DISPLAYSCREEN) printf("\n[Ray-Tracing]\n");

        RenderCalculation(x, y, z, psi, theta, phi);
        distanceAuvLight = sqrt ((x-Light_Source_center_Z*feet_per_meter)
```

```
                    (x-Light_Source_center_Z*feet_per_meter)+ (y-
                    Light_Source_center_X*feet_per_meter) *(y-
                    Light_Source_center_X*feet_per_meter)+(z+Light_Source_center_
                    Y*feet_per_meter) *(z+Light_Source_center_Y*feet_per_meter));

               printf("distanceAuvLight=%f\n",distanceAuvLight);
               if (CalculateCspeedDpsiDz(&CSpeed, &DeltaPsi, &DeltaZ) &&
                                                        (distanceAuvLight<6)
                                                        )
               {
                         FOLLOWLIGHTCONTROL = FALSE;
                         if (DISPLAYSCREEN) printf("\n[end followlight ]");
               }

               DEADSTICKPLANES = TRUE;
               DEADSTICKRUDDER   = FALSE;
               TARGETPOINTING    = FALSE;
               REPORTSTABLE      = TRUE;
               port_rpm_command   = 400*CSpeed;
               stbd_rpm_command   = 400*CSpeed;
               time_int_control_on  = t + 10.0; /* give PD control 10 seconds */
               z_command  = z + DeltaZ;
               compute_lateral_thrusters ();
               psi_command = normalize ((normalize2(psi + DeltaPsi)));
               psi_command_hover = psi_command;
               rotate_command    = 0.0;
               lateral_command   = 0.0;

                 }

               time_next_command = t + 2.0 * dt;
               if ((TRACE || TRUE) && DISPLAYSCREEN)
                     printf ("[complete compute_followlight]\n");
}
/* end compute_followlight_controls () */


/*************************************************************************/
```

5. asty_static.cpp

```
//--------------------------------------------------------------------------
----
//
//
//  Revised:          3 July 2000
//
//  SUBSYSTEM:    Astyanax-AUV
//  FILE:         Asty_static.cpp
//  AUTHOR:       Eric Hospital / Xavier Dussourd
//
//  OVERVIEW
//  ~~~~~~~~
//  Vision module functions
//
//--------------------------------------------------------------------------
----
#include "Frame.h"
#include <stdio.h>
#include "..\execution\statevector.h"
#include "..\execution\globals.h."
extern "C"
{
   void RenderInitialization();
   void RenderFinish();
   void RenderCalculation(double x, double y, double z, double phy, double
theta, double psy);
   int  CalculateCspeedDpsiDz(double *cs, double *psy, double *zz);
   double CalculatePsiLightReference();
   void SaveImage();
}


  FILE *followlightFile;
  static CFrame MyFrame;
  static unsigned Ym = 1;
  static unsigned Xm = 1;
  static int ParsedFlag = 0;
  static double feet_per_meter = 3.28084; /* number of feet per meter*/

  static double cameraX, cameraY, cameraZ; /*position of the camera in the
worlds sytem*/
  static double cameraU=(1*feet_per_meter); /*position of the camera about
u (body longitudinal axis) in feet*/
  static char nameImageFile[50];

void RenderInitialization()
{
   followlightFile=fopen("../Astyanax/followlight","w");
```

```
   fprintf(followlightFile, "  t    xg    yg   psi    dz    cs      d
distanceAuvLight(feet)\n");
   printf("\n[Creating images subdirectory (if needed)]\n");
/*   int result = system ("mkdir ../Astyanax/images");    not working  :(
*/
   printf("\n[Parsing Rendering Script]\n");
   ParsedFlag=1;
   MyFrame.Destroy();
   MyFrame.Parse("../Astyanax/Astyanax.scn");
   Xm = MyFrame.Screen_Width;
   Ym = MyFrame.Screen_Height;
}

void RenderFinish()
{
   MyFrame.Destroy();
   fclose(followlightFile);
}

void RenderCalculation(double rx, double ry, double rz, double rpsi,
double rtheta, double rphi)
{
   if(!ParsedFlag)
      RenderInitialization();

      printf("Locationt (%5.11f,%5.11f,%5.11f) PsiThetaPhi
(%5.11f,%5.11f,%5.11f)\n", rx, ry, rz, rpsi, rtheta, rphi);

      cameraX = rx + cameraU*cos(rtheta*M_PI/180.0)*cos(rpsi*M_PI/180.0);
      cameraY = ry + cameraU*cos(rtheta*M_PI/180.0)*sin(rpsi*M_PI/180.0);
      cameraZ = rz - cameraU*sin(rtheta*M_PI/180.0);

      MyFrame.Camera.Location = CVecteur(cameraY/feet_per_meter, -
cameraZ/feet_per_meter, cameraX/feet_per_meter);
      CTransform TheTrans;

      TheTrans.Compute_Passage(rpsi*M_PI/180.0, rtheta*M_PI/180.0,
rphi*M_PI/180.0);

      const CVecteur
DirReel(TheTrans.InvTransDirection(CVecteur(1.0,0.0,0.0)));
      MyFrame.Camera.Direction = CVecteur (DirReel.Y, -DirReel.Z,
DirReel.X);

      const CVecteur RightReel(TheTrans.InvTransDirection(CVecteur(0.0,
1.0, 0.0)));
      MyFrame.Camera.Right = CVecteur (RightReel.Y, -RightReel.Z,
RightReel.X);

      const CVecteur UpReel(TheTrans.InvTransDirection(CVecteur(0.0,0.0,-
1)));
      MyFrame.Camera.Up= CVecteur (UpReel.Y, -UpReel.Z, UpReel.X);

      MyFrame.Start_Tracing();
   }

   int CalculateCspeedDpsiDz(double * cs, double * dpsi, double * dz)
```

```
{
     sprintf(nameImageFile,"../Astyanax/images/Image%04.0f.bmp",t*10);
     printf("nameImageFile=%s\n", nameImageFile);
   if(MyFrame.Image)
   {
          MyFrame.Save(nameImageFile);
        CIntensityPoint ip;
        MyFrame.Image->GetIPointNearBary(&ip);
        const double yg = ip.bcYNear;
        const double xg = ip.bcXNear;
        const double d = sqrt(((2.0*yg-Ym)*(2.0*yg-Ym)+(2.0*xg-
        Xm)*(2.0*xg-Xm))/(Xm*Xm+Ym*Ym));
        printf("xg= %5.11f  yg= %5.11f  d=%5.11f\n", xg, yg, d);
        if (xg==Xm/2) printf("HEADING OK\n");


        * dpsi = atan2(2.0*xg-Xm,1.0*Xm)*180.0/M_PI;

        if ((yg>=(Ym/2)+(Ym/20)) || (yg<=(Ym/2)-(Ym/20)))
          {
              printf ("grand DZ\n");
              * dz=10*(2*yg-Ym)/Ym;
          }
        else * dz=(2*yg-Ym)/Ym;

        * cs = 0.5*(exp(-(xg-(Xm/2))*(xg-(Xm/2))/400)+exp(-(yg-
        (Ym/2))*(yg-Ym/2)/400));

        printf ("Astyanax calculated Cspeed,Psi,Z (%5.11f, %5.11f,
        %5.11f)\n", * cs, * dpsi, * dz);

        fprintf (followlightFile, "%5.11f %5.11f %5.11f %5.11f %5.11f
        %5.11f %5.11f %5.11f\n", t, xg, yg, * dpsi, * dz, * cs, d,
        distanceAuvLight);
        if (d<=0.06) {return 1;}
   }
   return 0;
}


   double CalculatePsiLightReference()
     {
        static double psiRef =0;
        static double maxIntensityRef =0;
        static double dRef = 0;
        static int firstLoopFlag =0;

        if (firstLoopFlag == 0)
        {
          psiRef = psi;
          firstLoopFlag   = 1;
        }

        sprintf(nameImageFile,"../Astyanax/images/Image%04.0f.bmp",t*10);
          if(MyFrame.Image)
        {
              MyFrame.Save(nameImageFile);
```

```
                    CIntensityPoint ip;
                    MyFrame.Image->GetIPointNearBary(&ip);
                    const double yg = ip.bcYNear;
                    const double xg = ip.bcXNear;
                    double maxIntensity = ip.bcIntensity;
                    double d = sqrt(((2.0*yg-Ym)*(2.0*yg-Ym)+(2.0*xg-
                    Xm)*(2.0*xg-Xm))/(Xm*Xm+Ym*Ym));

                    printf ("maxIntensity=%f\n",maxIntensity);

                    if (maxIntensity > maxIntensityRef)
                    {
                          psiRef = psi;
                          maxIntensityRef = maxIntensity;
                          dRef = d;
                    }

                    if (maxIntensity == maxIntensityRef)
                    {
                          if (d<dRef)
                          {
                            psiRef = psi;
                            maxIntensityRef = maxIntensity;
                            dRef = d;
                          }
                    }
              }

        printf("psiRef=%5.3lf  maxIntensityRef=%5.3lf
        dRef=%5.3lf\n",psiRef, maxIntensityRef, dRef);
        return (psiRef);
         }


void SaveImage()
  {
        sprintf(nameImageFile,"../Astyanax/images/Image%04.0f.bmp",t*10);
        if(MyFrame.Image)
        MyFrame.Save(nameImageFile);
  }
```

APPENDIX D

This appendix provides an example of the files needed to run a recovery mission simulation.

1. **`Astyanax.scn`**
2. **`Astyanax.scn.HELP`**
3. **`mission.script.followlight`**
4. **`TelemetryPlaybackWithFollowlightSubmarine.java`**
5. **`AuvInBeachTanksWithFollowlightSubmarine.wrl`**
6. **Launch followlight mission procedure**

1. **Astyanax.scn**
   This file describes the scene that Astyanax has to render.

```
/*************************************************************************/
//file: astyanax.scn
//
//author Xavier Dussourd
//
//purpose: scene with a submarine, a light_source, a black background_color
//and a color camera
//
//coordinates in meter
/*************************************************************************
*/

camera = {
    height = 96;
    width = 96;
};

atmosphere ={
    ior = 1.33;
    scattering = {
         color = green 0.3;
         distance =10;
         type = mie_hazy();
         samples = 50;
    };
};


object = { sphere(<22,-49.4,37>,1); scale(<15,5,5>); color = blue 1.0
                                                    green 1.0;};
object = { cylinder(<23,-49.4,37>,<-8,-49.4,37>, 5); color = blue 1.0
                                                    green 1.0;};
object = { cone(<-20,-49.4,37>,1, <-8,-49.4,37>,5); color = blue 1.0 green
                                                          1.0;};
object = { cone(<27,-50.4,32>,1.5, <36,-49.4,37>,0, open); color = blue
                                                  1.0 green 1.0;};
object = { cylinder(<22,-48.4,37>, <22,-49.4,37>, 1); scale(<4,8,2>);
                                          color = blue 1.0 green 1.0; };
light_source = {
     center = <28, -50, 32>;
     color = blue .5 green .6 red .5;
     };
```

2. astyanax.scn.HELP

```
//--------------------------------------------------------------------//

          astyanax.scene.HELP                                10 July 2000

          Syntax for describing Astyanax scene.

        Presents only the functions needed to describe underwater scene
for the followlight process.
        Complete syntax available in the Astyanax User's Guide [Caroline
Deltheil 1997]


   Xavier Dussourd                                  xdussour@nps.navy.mil

//--------------------------------------------------------------------//

Where describe a scene
      Astyanax scene has to be describe in the Astyanax.scn file.
      The units used are meter and radians.

How describe a scene:

Comment notation
      Astyanax comments :
      start with /* and end with */. They can include carriage returns.
      or start with // and terminate with a carriage return

Decimal numbers
      Astyanax accepts the following syntax decimal numbers :
      1.0   -2.0   -4   34   3.7e6   2.5e-5    .3   0.6

Mathematical expressions
      Astyanax accepts the following simple mathematical expressions :
      1+2+3.0   2*5   1/3   (7-2)/5

Syntax Vectors
      The vectors belong  to the Astyanax  3D coordinate system x, y and
z. They are written by  three expressions of decimal numbers between  <
and > and separated by commas.
      <x, y, z>

Colors
      Colors are expressed in the standard screen rgb corrdinates (value
between 0 and 1). They are introduced by the rgb keyword followed by three
expressions of decimal numbers put between brackets and separated by
commas :
      Example : Rgb (1.0, 0, 0.0)

Objects
      Objects compose a scene. They are introduced by the object keyword
followed by a lock describing the object.

      object = {" <type_objet> [<i_modifs>]"};
```

Differents objects

Astyanax can read 5 type_objet (box, sphere, cone, cylinder and plane).

All the objects execped plane can be inverted and become their complementary by applying the keyword invert().

The keyword box introduces rectangle parallelepipeds Between brackets should follow two vectors being the edge of the box.

```
object  = { box (  <0, 0, 0>, <1, 1, 1> ); };
```

Each element of the first vector has to be smaller than the element corresponding to the second vector; if it is not the case, the parallepiped will not appear

The keyword sphere introduces spheroids. Between brackets should follow the vector of the position of the spheroid center and its raduis :

```
object  = { sphere (  <0, 0, 0>, 3.9 ); };
```

The keyword cone introduces cones. Between brackets should follow the vector of the position of the center of one of the cone sides, the raduis of this side, a vector of the center of the other side of the cone and its raduis :

```
object  = { cone (  <0, 0, 0>,  2.0 , <1, 1, 1>, 1.0 ); };
```

The ends of the cone can be open if added the keyword open :

```
object  = { cone (  <0, 0, 0>,  2.0 , <0, 3, 0>, 0, open ); };
```

The keyword cylinder introduces circular cylinders. Between brackets should follow the vector of the position of the two ends of the cylinder and a decimal number for the cylinder raduis.

```
object  = { cylinder (  <0, 0, 0>, <3, 0, 0>, 7.2 ); };
```

The ends of the cone can be open if added the keyword open :

```
object  = { cylinder (  <0, 0, 0>, <3, 0, 0>, 7.2, open ); };
```

The keyword plane introduces planes. Between brackets should follow the vector for the direction of the normal to the plane, and the distance of the plane to the origin :

```
object  = { plane (  <0, 1, 0>,  4 ); };
```

The plane of equation A*x + B*y + C*z = D is for Astyanax :

```
object  = { plane (  <A, B, C>,  D ); };
```

Compositons

Objects additions are done by adding objects.

Example of a cylinder with ronded sides :

```
object  = { sphere (  <-2, 0, 3>, 1 ); };
object  = { cylinder (  <-2, 0, 3>,   <2, 0, 3>, 1 ); };
object  = { sphere (  <2, 0, 3>, 1 ); };
```

Subtractions are done by adding the ketword clippers to the object description block :

Example of a portion of a sphere :

```
object  = {
sphere (  <0, 0, 0>, 1 ); };
clippers =  {
      object = {
                  plane ( <1, 0 , 0>, 0);
```

```
        };
    };
```

Texture

Textures describe materials composing Astyanax objects. They detail the color, shade the effects and properties as the transparency or the brilliance.

The texture modifications will be included in the object desciption block.

Textures are made of three elements (the pigment or color set, the tnormal or form defect, the finish or surface optical properties).

Pigment

Mono color object

The keyword color will be used

Example of a blue sphere:

```
object = {
    sphere ( <0, 0, 0>, 1);
    color = rgb (0, 0, 1);
    };
```

Many colors object

Astyanax can simulated object with many colors. This part is available in the Astyanax User's Guide.

Finish

Finish describe the phisico-optical parameters of the surface. Below are presented the useful parameters (more parameters are available in the Astyanax User's Guide)

The keyword brillance introduced the control of the luminosity effect of the object

The keyword reflection introduced the reflection rate

The keyword refraction introduced the refraction rate

The keyword ior introduced the refraction coefficient

All these keywords should be followed by = and their value

Example

```
object = {
sphere ( <0, 0, 0>, 1);
color = rgb (0, 0, 1);
ior = 1.33;
reflection = 10;
};
```

Tnormal

Astyanax can simulated the form effects. This part is available in the Astyanax User's Guide.

Camera

The camera is introtuced by the keyword camera. In the description block follows

the keyword gray_camera or color_camera (default color_camera to verify)

the keyword width followed by the number of pixels in the width.

the keyword height followed by the number of pixels in the heigh

Example

```
camera = {
gray_camera;
height = 96;
      width = 96;
      };
```

Lights

Lights are introduced by the keyword light_source. In the description block follows the source position (default <0, 0, 0>), the source color and the keyword max_intensity_threshold (see modification).

Example :

```
light_source = {
center = <91.84, -16.4, 105>;
max_intensity_threshold = 100;
```

Atmosphere

The atmosphere description is introduced by the keyword atmosphere. It should be followed by :

The keyword ior decribing the refraction coefficient of the medium followed by is value

The block absorption introduced by the keyword absorption.

This block contains :

The keyword distance describing here the De Beer distance (63% of the light is absorbed at this distance).

The keyword color describing here the filter

The block scattering introduced by the keyword scatter.

This block contains :

The keyword distance describing here at 63% of the light is lost by diffussion at this distance).

The keyword color distributing the colors coming from this filter

The keyword samples describing the diffusion calculations accurancy

The keyword type describing the diffusion type ('mie_hazy' one modelizes the underwater diffusion)

Example :

```
atmosphere = {
ior = 1.33 ;
absorbtion = {
distance = 50;
color = rgb (0, 0, 0);
};
scatter = {
distance = 50;
color = rgb (1, 1, 1);
samples = 50 ;
type = mie_hazy ( );
};
```

```
};
```

Geometrical transformations

In blocks describing objects and lights, one can introduce keywords allowing translations, rotations, or homothtic transformation :

The keyword translate should be followed by the vector of translation applied to the object or the light in the Astyanax coordinates system.

Example :

```
light_source = {
center = <0 , 0 , 0>;
translate ( <10, 20, 2>);
};
```

The keyword rotate should be followed by the three rotations angles in degrees around the three axises (X, Y and Z) of the Astyanax coordinates system applied to the object.

Example

```
object = {
      box (<0 ,0, 0>, <1, 2, 3>);
      rotate = (<90, 0 ,0>);
};
```

The keyword scale introduced an homothetic transformation to the object. It should be followed by the three scale effects on the three axises (X, Y and Z) of the Astyanax coordinates system applied to the object.

Example of an ellipsoid center at the origin, raduis 1 for X and Y and raduis 4 for Z.

```
object = {
      sphere (<0 ,0, 0>, 1);
      scale = (<1, 1, 4>);
      };
```

7. mission.script.followlight

This file describes the mission of the AUV. It includes the 'followlight' keyword.

```
#mission of recovery between the AUV and a submarine
#using the followlight control
#feet

position -10 50 145
speed 200
thrusters-on
hover -10 40 145 200
wait 5
followlight
hover
wait 100
quit
```

4. AuvInBeachTanksWithFollowlightSubmarine.wrl

This code describes the VRML scene. Same scene that this one described in the `Astyanax.scn` file (objects and light). Code added to `AuvInBeachTanksWithFollowlightSubmarine.wrl`

Units: meter

```
/*************************************************************************/
            Transform {
                        translation 32 -50 28  # center of the light
                        children [
                              Viewpoint {
                                    description "light of followlight"
                                    orientation 1 0 0 0
                              }
                             ]
                        }

                  Transform {
                        translation 32 -50 28  # center of the light
                        children [
                              Shape {
                                    appearance Appearance {
                                          material Material{
                                          diffuseColor 0.5 0.6 0.5
                                          }
                                    }
                                    geometry Sphere {
                                          radius 0.2
                                          }
                               }
                              SpotLight {
                                    direction 0.0 0.0 -1.0
                                    radius    100.00
                                    intensity 5.0
```

```
      ambientIntensity 0.2
      color  0.5 0.6 0.5
      cutOffAngle  2.09
}
Transform {
            translation 5 0.6 -6
      scale 5.0 5.0 15.0
      children [
            Shape {
               appearance Appearance {
                  material Material{
                  diffuseColor 0.25 0.25 0.25
                  }
                }
               geometry Sphere {
                  radius 1
                    }
             }
      ]
    }
Transform {
      translation 5 0.6 -20.5
      rotation 1.0 0.0 0.0  1.570796
      children [
         Shape {
           appearance Appearance {
              material Material{
              diffuseColor 0.25 0.25 0.25
              }
             }
             geometry Cylinder {
                 radius 5
                 height 31
             }
                }
       ]
        }
Transform {
      translation 5 0.6 -42
      rotation -1.0 0.0 0.0  1.570796
      children [
            Shape {
                  appearance Appearance {
                        material Material{
                              diffuseColor
                              0.25 0.25 0.25
                        }
                  }
                  geometry Cone {
                        bottomRadius 5
                        height 12
                  }
            }
      ]
}
Transform {
            translation 0.609 0.0792 0.243
      rotation 1.0 0.0 0.0  1.570796
      children [
            Shape {
                  appearance Appearance {
                        material Material{
                              diffuseColor
                              0.25 0.25 0.25
```

```
                                                    }
                                                }
                                                geometry Cone {
                                                    bottomRadius 1.5
                                                    height 1
                                                    bottom FALSE
                                                }
                                            }
                                        ]
                                    }

                                ]
                            }

/*****************************************************************************
```

5. TelemetryPlaybackWithFollowlightSubmarine.java

This code describes VRML scene. Same scene that `Astyanax.scn` (light, objects). Code includes in `TelemetryPlaybackWithFollowlightSubmarine.java`. Used to play back the mission viewer.

```
System.out.println ("Group {");
System.out.println ("    children [\n")
System.out.println ("          Transform {");
System.out.println ("                translation 104.98688 -164.042 91.86352
                    # center of the light");
System.out.println ("                children [");
System.out.println ("                      Viewpoint {");
System.out.println ("                            description \"light of
                  followlight\"");
System.out.println ("                            orientation 1 0 0 0");
System.out.println ("                      }");
System.out.println ("                    ]");
System.out.println ("                }");

System.out.println ("          Transform {");
System.out.println ("                translation 104.98688 -164.042
                    91.86352  # center of the light");
System.out.println ("                children [");
System.out.println ("                      Shape {");
System.out.println ("                            appearance Appearance {");
System.out.println ("                            material Material{");
System.out.println ("                            diffuseColor 0.5 0.6 0.5");
System.out.println ("                            }");
System.out.println ("                       }");
System.out.println ("                       geometry Sphere {");
System.out.println ("                            radius 0.7");
System.out.println ("                            }");
System.out.println ("                       }");
System.out.println ("                      SpotLight {");
System.out.println ("                            direction 0.0 0.0 -1.0");
System.out.println ("                            radius    100.00");
System.out.println ("                            intensity 5.0");
System.out.println ("                            ambientIntensity 0.2");
System.out.println ("                            color  0.5 0.6 0.5");
System.out.println ("                            cutOffAngle  2.09");
System.out.println ("                      }");
System.out.println ("                      Transform {");
System.out.println ("                                  translation 16.4042
                    1.968504 -19.68504");
System.out.println ("                            scale 5.0 5.0 15.0");
System.out.println ("                      children [");
System.out.println ("                Shape {");
System.out.println ("                      appearance Appearance {");
System.out.println ("                      material Material{");
System.out.println ("                      diffuseColor 0.25 0.25 0.25");
System.out.println ("                            }");
System.out.println ("                      }");
System.out.println ("                      geometry Sphere {");
System.out.println ("                        radius 3.28084");
System.out.println ("                         }");
System.out.println ("                      }");
System.out.println ("                    ]");
System.out.println ("                }");
System.out.println ("          Transform {");
System.out.println ("                translation 16.4042 1.968504 -
67.25722");
System.out.println ("                rotation 1.0 0.0 0.0  1.570796");
System.out.println ("                children [ ");
System.out.println ("                      Shape {");
System.out.println ("                            appearance Appearance {");
System.out.println ("                            material Material{");
System.out.println ("                            diffuseColor 0.25 0.25
0.25");
System.out.println ("                            }");
```

```
System.out.println ("                          }");
System.out.println ("                         geometry Cylinder {");
System.out.println ("                           radius 16.4042");
System.out.println ("                           height 101.70604");
System.out.println ("                             }");
System.out.println ("                              }");
System.out.println ("                       ]");
System.out.println ("                 }");
System.out.println ("          Transform {");
System.out.println ("                 translation 16.4042 1.968504 -
                137.79528");
System.out.println ("                 rotation -1.0 0.0 0.0  1.570796");
System.out.println ("                 children [  ");
System.out.println ("                 Shape {");
System.out.println ("                       appearance Appearance {");
System.out.println ("                       material Material{");
System.out.println ("                       diffuseColor 0.25 0.25 0.25");

System.out.println ("                       }");
System.out.println ("                    }");
System.out.println ("                 geometry Cone {");
System.out.println ("                          bottomRadius 16.4042");
System.out.println ("                          height 39.37008");
System.out.println ("                            }");
System.out.println ("                           }");
System.out.println ("                        ]");
System.out.println ("          }");

System.out.println ("          Transform {");
System.out.println ("                       translation 2 0.26 0.8");
System.out.println ("                 rotation 1.0 0.0 0.0  1.570796");
System.out.println ("                 children [");
System.out.println ("                         Shape {");
System.out.println ("                         appearance Appearance {");
System.out.println ("                         material Material{");
System.out.println ("                          diffuseColor 0.25 0.25 0.25");

System.out.println ("                             }");
System.out.println ("                          }");
System.out.println ("                         geometry Cone {");
System.out.println ("                           bottomRadius 4.92126");
System.out.println ("                           height 3.28084");
System.out.println ("                         bottom FALSE");
System.out.println ("                            }");
System.out.println ("                           }");
System.out.println ("                     ]");
System.out.println ("                 }");
System.out.println ("      ]");
System.out.println ("}");
System.out.println ("");
```

6. Launch followlight mission procedure

To launch a followlight mission:

1. Run `dynamics` in a DosWindow `~dynamics> java dynamics loop`

2. Run `execution` in a DosWindow `~execution>execution remote` [computername] `mission followlight real-time`

3. To see the mission with the viewer in real time, run the file `AuvInBeachTanksWithFollowlightSubmarine.wrl` in Netscape with DIS-Java-VRML installed.

8. To compile the images in a video format `.avi` file run the invocation `C:\auv\astyanax\images> bmp2avi -i Image -o mission.followlight -f 10 -k 1` with the `bmp2avi.exe` program in the `C:\auv\astyanax\images` directory.

REFERENCES

Brutzman, Don, "Virtual World Visualization for an Autonomous Underwater Vehicle," Proceedings of the IEEE Oceanic Engineering Society Conference OCEANS 95, San Diego California, October 12-15 1995, pp. 1592-1600.
Available at *ftp://taurus.cs.nps.navy.mil/pub/auv/oceans95.ps.Z*

Burns, Michael L., *Merging Virtual and Real Execution Level Control Software for the Phoenix Autonomous Underwater Vehicle,* Thesis
Naval Postgraduate School, Monterey, CA, September 1996

Deltheil C., Hospital E., *Astyanax User's Guide*, January 1997

Deltheil C., Hospital E., *Astyanax Reference Guide*, January 1997

Deltheil, Caroline, Leandri, Didier, Hospital, Eric and Brutzman, Donald P., "Simulating an Optical Guidance System for the Recovery of an Unmanned Underwater Vehicle," *IEEE Journal of Oceanic Engineering*, to be published 2000.
Available at *web.nps.navy.mil/~auv/OpticalGuidanceSimulationJOE99.pdf*

Andrea L. Ames, David R. Nadeau, John L.Moreland , *The VRML 2.0 sourcebook*, 1997

Pov-Ray, the Persistence of Vision Raytracer
Available at http://www.povray.org

Decadenet [Turner, OR], BOB-II Technical Specifications
Available at http://www.decadenet.com

DeepSea [San Diego, CA], SST-1370 Camera Technical Specifications
Available at http://www.deepsea.com

Sony, *Protocol of Control L/LF*
Service Manual, March 1986

Sound Ocean Systems [Redmond, WA], *Instruction Manual for the Model 199-100-014 Logger Card*, 1999

Addenda Electronics [Monterey, CA], RS-4/L Control-L Interface Adapter Technical Specifications
Available at http://www.addenda-elect.com

First Witness [Mt Sydney, VA], Ultra Link Video Wireless Transmitter Description
Available at http://ultralinkvideo.com/

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....1
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library.....2
   Naval Postgraduate School
   Monterey, CA, 93943-5100

3. Ecole Nationale d'Ingenieurs de Tarbes (ENIT).....2
   Service Scolarite
   47, Avenue d'Azereix, BP 1629
   65016 Tarbes Cedex, France

4. Dr. Don Brutzman 1
   Undersea Warfare Academic Group, Code UW/Br
   Naval Postgraduate School
   Monterey, CA, 93943-5100

5. CAPT John C. Mickey USN.....1
   Team Submarine (PMS 4012B)
   Naval Sea Systems Command (NAVSEA)
   Arlington, Virginia 22217

6. Dr. Thomas Curtin.....1
   Office of Naval Research (ONR)
   800 North Quincy Street
   Arlington, Virginia 22217

7. Dr. Caroline Deltheil.....1
   Residence Pullman
   13 rue de l'Hourtoulane
   66000 Perpignan, France

8. Olivier Doucy.....1
   SIREHNA
   1 rue de la Noe
   BP 42105
   44321 Nantes, France

9. Xavier Dussourd.....1
54 rue de Lescure
33000 Bordeaux, France

10. Dr. Anthony J. Healey, Code ME/Hy.....1
Mechanical Engineering Department
Naval Postgraduate School
Monterey, CA, 93943-5100

11. Didier Leandri.....1
"Les Restanques du Thouar"
36 Impasse des Sorbiers
83130 La Garde, France

12. Jean-Pierre LeGoff 1
SIREHNA
1 rue de la Noe
BP 42105
44321 Nantes CEDEX 3, France

13. Fred Liddy.....1
Sonalysts Inc.
215 Parkway North
Waterford, CT 06385

14. Dr. David Marco, Code ME/Ma.....1
Mechanical Engineering Department
Naval Postgraduate School
Monterey, CA, 93943-5000

15. CAPT John C. Mickey USN.....1
PEO for Submarines (PMS401)
2531 Jefferson Davis Hwy
Arlington, VA 22242-5161

16. LCDR Jeffery S. Riedel, USN, Ph.D.....1
PEO TSC (PMS 400D4A)
2531 Jefferson Davis Highway
Arlington, VA 22242-5165

17. Dr. Samuel M. Smith.....1
Dept of Ocean Engineering
Florida Atlantic University
500 N.W. 20 Street
Boca Raton, FL 33431-0991

18. Dr. Thomas Swean.....1
Office Of Naval Research (ONR)
800 North Quincy Street
Arlington, VA 22217

19. Thomas N. Stewart.....1
The Johns Hopkins University
Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, MD. 20723-6099

20. Robert Stock.....1
Program Executive Office (PEO) for UnderSea Warfare (USW)
Advanced Systems and Technology Office (ASTO)
2531 Jefferson Davis Highway
Arlington VA 22242-5160

21. Gary Trimble, Program Manager.....1
EODRWP
Lockheed Martin Inc.
Organization 8K-01, Building 586E
1111 Lockeed Way
Sunnyvale, CA 94089-3504

22. Dr. Thierry Vidal.....1
Ecole Nationale d'Ingenieurs de Tarbes
47, Avenue d'Azereix
BP 1629
65016 Tarbes Cedex, France